VIII. Модуль shelve (*.ini)

Для роботи з бінарними файлами Python може застосовуватися ще один модуль - shelve. Він зберігає об'єкти у файл із певним ключем. Потім за цим ключем може витягти збережений об'єкт з файлу. Процес роботи з даними через модуль shelve нагадує роботу зі словниками, які також використовують ключі для збереження та вилучення об'єктів.

Для відкриття файлу модуль shelve використовує функцію open():

```
open(шлях_до_файлу[, flag="c"[, protocol=None[,
writeback=False]]])
```

Де параметр flag може приймати значення:

- **c**: файл відкривається для читання та запису (значення за промовчанням). Якщо файл не існує, він створюється.
- **r**: файл відкривається лише для читання.
- **w**: файл відкривається для запису
- **n**: файл відкривається для запису Якщо файл не існує, він створюється. Якщо він існує, то він перезаписується

Для закриття підключення до файлу викликається метод close():

```
importshelve
d = shelve.open(filename)
d.close()
```

Або можна відкривати файл за допомогою оператора with.

Збережемо та рахуємо у файл кілька об'єктів:

```
importshelve

FILENAME = "states2"
withshelve.open(FILENAME) as states:
    states["London"] = "Great Britain"
    states["Paris"] = "France"
    states["Berlin"] = "Німеччина"
    states["Madrid"] = "Spain"

withshelve.open(FILENAME) as states:
    print(states["London"])
    print(states["Madrid"])
```

Запис даних передбачає встановлення значення для певного ключа:

```
states["London"] = "Great Britain"
```

А читання з файлу еквівалентне отриманню значення за ключом:

```
print(states["London"])
```

Як ключі використовуються рядкові значення.

При читанні даних, якщо запитуваний ключ відсутній, то генерується виняток. У цьому випадку перед отриманням ми можемо перевіряти наявність ключа за допомогою оператора in:

```
withshelve.open(FILENAME) as states:
    key = "Brussels"
    if key in states:
        print(states[key])
```

Можна також використовувати метод get(). Перший параметр методу - ключ, яким слід отримати значення, а другий - значення за промовчанням, яке повертається, якщо ключ не знайдено.

```
withshelve.open(FILENAME) as states:
    state = states.get("Brussels", "Undefined")
    print(state)
```

Використовуючи цикл for, можна перебрати всі значення файлу:

```
withshelve.open(FILENAME) as states:
    for key in states:
        print(key," - ", states[key])
```

Метод keys() повертає всі ключі з файлу, а метод values() - всі значення:

```
withshelve.open(FILENAME) as states:

    for city in states.keys():
        print(city, end=" ") # London Paris Berlin
Madrid
    print()
    for country in states.values():
        print(country, end=" ") # Great Britain #
France Germany Spain
```

Ще один метод items() повертає набір кортежів. Кожен кортеж містить ключ та значення.

```
with shelve.open(FILENAME) as states:

    for state in states.items():
        print(state)
```

Консольний висновок:

```
("London", "Great Britain")
("Paris", "France")
("Berlin", "Німеччина")
("Madrid", "Spain")
```

Оновлення даних

Для зміни даних достатньо присвоїти по ключу нове значення, а для додавання даних визначити новий ключ:

```
importshelve

FILENAME = "states2"
withshelve.open(FILENAME) as states:
    states["London"] = "Great Britain"
    states["Paris"] = "France"
    states["Berlin"] = "Німеччина"
    states["Madrid"] = "Spain"

withshelve.open(FILENAME) as states:

    states["London"] = "United Kingdom"
    states["Brussels"] = "Belgium"
    for key in states:
        print(key, "-", states[key])
```

Видалення даних

Для видалення з одночасним отриманням можна використовувати функцію pop(), в яку передається ключ елемента та значення за промовчанням, якщо ключ не знайдено:

```
withshelve.open(FILENAME) as states:

    state = states.pop("London", "NotFound")
    print(state)
```

Також для видалення може застосовуватись оператор del:

```
withshelve.open(FILENAME) as states:

    del states["Madrid"] # видаляємо об'єкт # з ключем
Madrid
```

Для видалення всіх елементів можна використовувати метод clear():

```
with shelve.open(FILENAME) as states:

    states.clear()
```

IX. Модуль OS та робота з файловою системою

Ряд можливостей роботи з каталогами і файлами надає вбудований модуль os. Він містить багато функцій, нижче розглянуті лише основні з них:

- mkdir(): створює нову папку
- rmdir(): видаляє папку
- rename(): перейменовує файл
- remove(): видаляє файл

Створення та видалення папки

Для створення папки застосовується функція mkdir(), в яку передається шлях до створюваної папки:

```
importos

шлях щодо поточного скрипту
os.mkdir("hello")
абсолютний шлях
os.mkdir("c://somedir")
os.mkdir("c://somedir/hello")
```

Для видалення папки використовується функція rmdir(), в яку передається шлях до папки, що видаляється:

```
importos

шлях щодо поточного скрипту
os.rmdir("hello")
абсолютний шлях
os.rmdir("c://somedir/hello")
```

Перейменування файлу

Для перейменування викликається функція rename(source, target), перший параметр якої шлях до вихідного файлу, а другий - нове ім'я файлу. Як шляхи можуть використовуватися як абсолютні, так і відносні.

Наприклад, нехай у папці C://SomeDir/ розміщується файл somefile.txt. Перейменуємо його на файл "hello.txt":

```
importos

os.rename("C://SomeDir/somefile.txt",
"C://SomeDir/hello.txt")
```

Видалення файлу

Для видалення викликається функція remove(), яку передається шлях до файлу:

```
importos

os.remove("C://SomeDir/hello.txt")
```

Існування файлу

Якщо спробувати відкрити файл, який немає, то Python «викине» виняток FileNotFoundError.

Для вилову виключення можна використовувати конструкцію try...except. Однак можна вже до відкриття файлу перевірити, чи існує він за допомогою методу os.path.exists(path).

У цей метод передається повне ім'я файлу (шлях та ім'я), існування якого необхідно перевірити:

```
filename = input("Введіть шлях до файлу: ")
ifos.path.exists(filename):
    print("Вказаний файл існує")
else:
    print("Файл не існує")
```

Робота з операційною системою

Модуль os також надає безліч функцій для роботи з операційною системою, причому їхня поведінка, як правило, не залежить від ОС, тому програми залишаються переносимими. Тут будуть наведені найчастіше використовувані з них (деякі функції цього модуля підтримуються не всіма ОС).

os.name- Ім'я операційної системи. Доступні варіанти: 'posix', 'nt', 'mac', 'os2', 'ce', 'java'.

os.environ- словник змінних оточення. Змінюваний (можна додавати та видаляти змінні оточення).

os.getlogin() - Ім'я користувача, що увійшов до терміналу (Unix).

os.getpid() – поточний id процесу.

os.uname() - інформація про ОС. Повертає об'єкт з атрибутами: sysname – ім'я операційної системи, nodename – ім'я машини в мережі (визначається реалізацією), release – реліз, version – версія, machine – ідентифікатор машини.

os.access(path, mode, *, dir_fd=None, effective_ids=False, follow_symlinks=True) – перевірка доступу до об'єкта у поточного користувача. Прапори: os.F_OK – об'єкт існує, os.R_OK – доступний на читання, os.W_OK – доступний на запис, os.X_OK – доступний на виконання.

os.chdir(path) – зміна поточної директорії.

os.chmod(path, mode, *, dir_fd=None, follow_symlinks=True) – зміна прав доступу до об'єкта (mode – вісімкове число).

os.chown(path, uid, gid, *, dir_fd=None, follow_symlinks=True) - змінює id власника та групи (Unix).

os.getcwd() – поточна робоча директорія.

os.link(src, dst, *, src_dir_fd=None, dst_dir_fd=None, follow_symlinks=True) - створює жорстке посилання.

os.listdir(path=".") - список файлів та директорій у папці.

os.mkdir(Path, mode = 0o777, *, dir_fd = None) - створює директорію. OSError, якщо директорія існує.

os.makedirs(Path, mode = 0o777, exist_ok = False) - створює директорію, створюючи при цьому проміжні директорії.

os.remove(path, *, dir_fd=None) – видаляє шлях до файлу.

os.rename(src, dst, *, src_dir_fd=None, dst_dir_fd=None) - перейменовує файл або директорію з src на dst.

os.renames(old, new) - перейменовує old на new, створюючи проміжні директорії.

os.replace(src, dst, *, src_dir_fd=None, dst_dir_fd=None) - перейменовує з src на dst з примусовою заміною.

os.rmdir(Path, *, dir_fd = None) - видаляє порожню директорію.

os.removedirs(path)- Видаляє директорію, потім намагається видалити батьківські директорії, і видаляє їх рекурсивно, поки вони порожні.

os.symlink(source, link_name, target_is_directory = False, *, dir_fd = None) - Створює символічне посилання на об'єкт.

os.sync() – записує всі дані на диск (Unix).

os.truncate(path, length) – обрізає файл до довжини length.

os.utime(path, times=None, *, ns=None, dir_fd=None, follow_symlinks=True) - модифікація часу останнього доступу та зміни файлу. Або times - кортеж (час доступу на секундах, час зміни на секундах), чи ns - кортеж (час доступу на наносекундах, час зміни на наносекундах).

os.walk(top, topdown = True, on error = None, followlinks = False) - генерація імен файлів у дереві каталогів, зверху вниз (якщо topdown дорівнює True), або знизу вгору (якщо False). Для кожного каталогу функція walk повертає кортеж (шлях до каталогу, список каталогів, список файлів).

приклад– вивести на консоль список усіх *.txt файлів, розміщених у папці l:\dist та її підпапках:

```
importos
fromos.path import join
for(root, dirs, files) in os.walk('l:\\dist'):
      for filename in files:
            if filename.endswith('.txt'):
                  thefile = os. path.join(root, filename)
                  print( os.path.getsize(thefile), thefile)
```

os.system(command) - виконує системну команду, повертає код її завершення (у разі успіху 0).

os.urandom(n) – n випадкових байт. Можливе використання цієї функції у криптографічних цілях.

X. Приклад команд роботи з файлами та файловою системою

Розглянемо 8 вкрай важливих команд для роботи з файлами, папками і файловою системою в цілому.

Показати поточний каталог

Найпростіша і водночас одна з найважливіших команд для Python-розробника. Вона потрібна тому, що найчастіше розробники мають справу із відносними шляхами. Але в деяких випадках важливо знати, де ми знаходимося.

Відносний шлях хороший тим, що працює для всіх користувачів з будь-якими системами, кількістю дисків і так далі.

Так ось, щоб показати поточний каталог використовуємо вбудовану в Python OS-бібліотеку:

```
import os
os.getcwd()
```

Майте на увазі, що шлях, що повертається, є абсолютним.

Перевіряємо, чи існує файл або каталог

Перш ніж задіяти команду створення файлу або каталогу, варто переконатися, що аналогічних елементів немає. Це допоможе уникнути ряду помилок під час роботи програми, включаючи перезапис існуючих елементів з даними.

Функція os.path.exists () приймає аргумент рядкового типу, який може бути або ім'ям каталогу або файлом.

Перевіримо, чи існує каталог sample_data:

```
os.path.exists('sample_data')
```

```
[3]  os.path.exists('sample_data')

     True
```

Ця ж команда підходить і для роботи з файлами:

```
os.path.exists('sample_data/README.md')
```

```
[4]  os.path.exists('sample_data/README.md')

     True
```

Якщо папки чи файлу немає, команда повертає false.

```
[5]  os.path.exists('foobar')

     False
```

Об'єднання компонентів шляху

У попередньому прикладі використовувався сліш "/" для роздільника компонентів колії. У принципі, це нормально, але не рекомендується. Якщо

ви хочете, щоб ваша програма була крос платформною, такий варіант не підходить. Так, деякі старі версії ОС Windows розпізнають лише слєш "\" як роздільник.

Python вирішує цю проблему завдяки функції os.path.join().

Давайте перепишемо варіант із прикладу в попередньому пункті, використовуючи цю функцію:

```
os.path.exists(os.path.join('sample_data',
'README.md'))
```

```
[7]  os.path.exists(os.path.join('sample_data', 'README.md'))

     True
```

Створення директорії

Створимо директорію з ім'ям test_dir усередині поточної директорії. Для цього можна використовувати функцію os.mkdir():

```
os.mkdir('test_dir')
```

Погляньмо, як це працює на практиці.

```
[9]  print(f"test_dir existing: {os.path.exists('test_dir')}")
     os.mkdir('test_dir')
     print(f"test_dir existing: {os.path.exists('test_dir')}")

     test_dir existing: False
     test_dir existing: True
```

Якщо ми спробуємо створити каталог, який вже існує, то отримаємо виняток.

```
[11] os.mkdir('test_dir')

     ---------------------------------------------------------------------------
     FileExistsError                           Traceback (most recent call last)
     <ipython-input-11-48d7b58469aa> in <module>()
     ----> 1 os.mkdir('test_dir')

     FileExistsError: [Errno 17] File exists: 'test_dir'
```

Саме тому рекомендується завжди перевіряти наявність каталогу з певною назвою перед створенням нового:

```
if not os.path.exists('test_dir'):
    os.mkdir('test_dir')
```

Ще одна порада щодо створення каталогів. Іноді нам потрібно створити підкаталоги з рівнем вкладеності 2 або більше. Якщо ми досі використовуємо os.mkdir(), нам потрібно буде зробити це кілька разів.

І тут ми можемо використовувати os.makedirs(). Ця функція створить усі проміжні каталоги:

```
os.makedirs(os.path.join('test_dir', 'level_1',
'level_2', 'level_3'))
```

Ось що виходить у результаті.

Показуємо вміст директорії

Ще одна корисна команда – os.listdir(). Вона показує весь вміст каталогу.

Команда відрізняється від os.walk(), де останній рекурсивно показує все, що знаходиться під каталогом. os.listdir () набагато простіше у використанні, тому що просто повертає список вмісту:

```
os.listdir('sample_data')
```

```
[13] os.listdir('sample_data')

     ['README.md',
      'anscombe.json',
      'california_housing_train.csv',
      'california_housing_test.csv',
      'mnist_train_small.csv',
      'mnist_test.csv']
```

У деяких випадках потрібно щось більш просунуте, наприклад, пошук усіх CSV-файлів у каталозі «sample_data». У цьому випадку найпростіший спосіб - використовувати вбудовану бібліотеку glob (див. нижче):

```
з glob import globlist(glob(os.path.join('sample_data',
'*.csv'))))
```

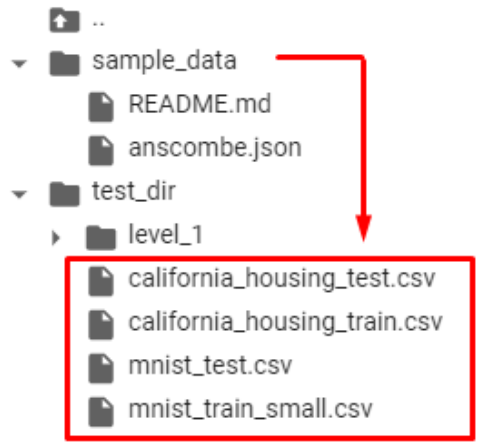
```
[14] from glob import glob

[15] list(glob(os.path.join('sample_data', '*.csv')))

     ['sample_data/california_housing_train.csv',
      'sample_data/california_housing_test.csv',
      'sample_data/mnist_train_small.csv',
      'sample_data/mnist_test.csv']
```

Переміщення файлів

Саме час спробувати перемістити файли з однієї папки до іншої. Рекомендований спосіб – ще одна вбудована бібліотека shutil (див. нижче).

Перемістимо всі CSV-файли з директорії sample_data до директорії test_dir:

```
import shutil for file in
list(glob(os.path.join('sample_data', '*.csv'))):

    shutil.move(file, 'test_dir')
```



До речі, є два способи виконати задумане. Наприклад, ми можемо використовувати бібліотеку OS, якщо не хочеться імпортувати додаткові бібліотеки. Як os.rename, і os.replace підходять вирішення завдання.

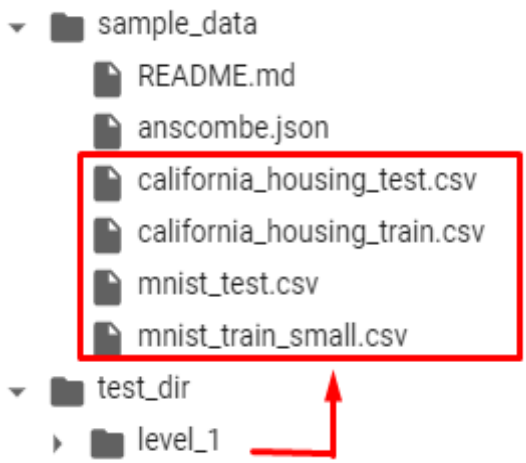
Але обидві вони недостатньо «розумні», щоб дозволити перемістити файли в каталог.

```
[21] for file in list(glob(os.path.join('test_dir', '*.csv'))):
         os.rename(file, 'sample_data')

---------------------------------------------------------------------------
IsADirectoryError                         Traceback (most recent call last)
<ipython-input-21-9675c94bbcdd> in <module>()
      1 for file in list(glob(os.path.join('test_dir', '*.csv'))):
----> 2     os.rename(file, 'sample_data')

IsADirectoryError: [Errno 21] Is a directory: 'test_dir/california_housing_train.csv' -> 'sample_data'
```

Щоб усе це працювало, потрібно явно вказати ім'я файлу на місці призначення. Нижче код, який це дозволяє зробити:

```
for file in list(glob(os.path.join('test_dir',
'*.csv'))):
    os.rename(
        file,
        os.path.join(
            'sample_data',
            os.path.basename(file)
```

```
) )
```



Тут функція os.path.basename() призначена для вилучення імені файлу зі шляху з будь-якою кількістю компонентів.

Інша функція, os.replace(), робить те саме. Але різниця в тому, що os.replace() не залежить від платформи, тоді як os.rename() працюватиме лише в системі Unix/Linux.

Ще один мінус у тому, що обидві функції не підтримують переміщення файлів з різних файлових систем, на відміну від shutil.

Тому рекомендується використовувати shutil.move() для переміщення файлів.

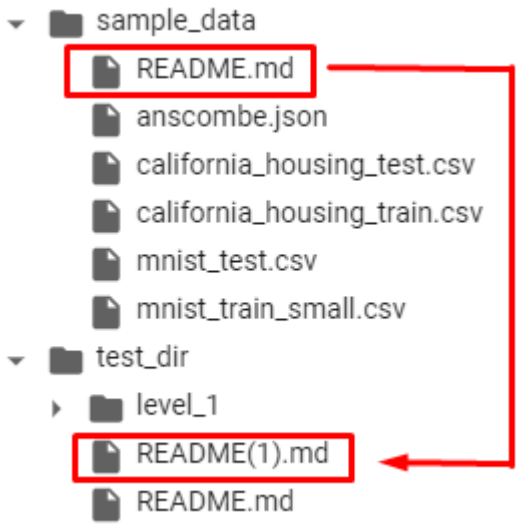
Копіювання файлів

Аналогічно shutil підходить і для копіювання файлів з вже згаданих причин.

Якщо потрібно скопіювати файл README.md з папки sample_data в папку test_dir, допоможе функція shutil.copy():

```
shutil.copy(
    os.path.join('sample_data', 'README.md'),
    os.path.join('test_dir')
)
```

```
[26] shutil.copy(
         os.path.join('sample_data', 'README.md'),
         os.path.join('test_dir', 'README(1).md')
     )

     'test_dir/README(1).md'
```



Видалення файлів та папок

Коли потрібно видалити файл, потрібно скористатися командою os.remove():

```
os.remove(os.path.join('test_dir', 'README(1).md'))
```

Якщо потрібно видалити каталог, використовуємо os.rmdir():

```
os.rmdir(os.path.join('test_dir', 'level_1', 'level_2',
'level_3'))
```

Однак os.rmdir() може видалити лише порожній каталог.

На наведеному рис. бачимо, що можна видалити лише каталог level_3.

Що, якщо необхідно рекурсивно видалити каталог level_1? У цьому випадку використовуємо shutil.

```
[28] os.rmdir(os.path.join('test_dir', 'level_1'))

---------------------------------------------------------------------------
OSError                                   Traceback (most recent call last)
<ipython-input-28-352a126b9aab> in <module>()
----> 1 os.rmdir(os.path.join('test_dir', 'level_1'))

OSError: [Errno 39] Directory not empty: 'test_dir/level_1'
```

Функція shutil.rmtree() зробить все, що потрібно:

```
shutil.rmtree(os.path.join('test_dir', 'level_1'))
```

Користуватися нею потрібно обережно, оскільки вона безповоротно видаляє весь вміст каталогу.

XI. Модуль shutil

Модуль shutil містить набір функцій високого рівня обробки файлів, груп файлів, і папок. Зокрема, доступні тут функції дозволяють копіювати, переміщати та видаляти файли та папки. Часто використовується разом із модулемos.

Операції над файлами та директоріями

shutil.copyfileobj(fsrc, fdst [, length])

- Скопіювати вміст одного файлового об'єкта (fsrc) в інший (fdst).

Необов'язковий параметр length - розмір буфера при копіюванні (щоб весь, можливо величезний, файл не читався повністю на згадку).

При цьому, якщо позиція покажчика fsrc не 0 (тобто до цього було зроблено щось на зразок fsrc.read(47)), то буде копіюватися вміст починаючи з поточної позиції, а не з початку файлу.

shutil.copyfile(src, dst, follow_symlinks = True)

- Копіює вміст (але не метадані) файлу src у файл dst.

Повертає dst (тобто куди файл було скопійовано). src та dst це рядки - шляхи до файлів. dst має бути повним ім'ям файлу.

Якщо src і dst є один і той же файл, виключення shutil.SameFileError.

Якщо dst існує, він буде перезаписаний.

Якщо follow_symlinks=False та src є посиланням на файл, то буде створено нове символічне посилання замість копіювання файлу, на який це символічне посилання вказує.

shutil.copymode(src, dst, follow_symlinks = True)

- Копіює права доступу з src в dst. Вміст файлу, власник і група не змінюються.

shutil.copystat(src, dst, follow_symlinks = True)

- копіює права доступу, час останнього доступу, останньої зміни та прапори src в dst. Вміст файлу, власник і група не змінюються.

shutil.copy(src, dst, follow_symlinks=True) - копіює вміст файлу src у файл чи папку dst. Якщо dst є директорією, файл буде скопійовано з тією

самою назвою, що була у src. Функція повертає шлях до розташування нового скопійованого файлу.

Якщо follow_symlinks=False і src це посилання, dst буде посиланням.

Якщо follow_symlinks=True, і src це посилання, dst буде копією файлу, який посилається src

copy() копіює вміст файлу та права доступу.

shutil.copy2(src, dst, follow_symlinks = True)

- як copy(), але намагається копіювати всі метадані.

shutil.copytree(src, dst, symlinks=False, ignore=None, copy_function=copy2, ignore_dangling_symlinks=False)

- рекурсивно копіює все дерево директорій з коренем у src, повертає директорію призначення.

Директорія dst має існувати. Вона буде створена разом із пропущеними батьківськими директоріями.

Права та часи у директорій копіюються copystat(), файли копіюються за допомогою функції copy_function (за замовчуванням shutil.copy2()).

Якщо symlinks=True, посилання в дереві src будуть посиланнями в dst і метадані будуть скопійовані настільки, наскільки це можливо.

Якщо False (за промовчанням), буде скопійовано вміст і метадані файлів, на які вказували посилання.

Якщо symlinks=False, якщо файл, на який вказує посилання, немає, буде додано виняток до списку помилок, у виключенні shutil.Error в кінці копіювання.

Можна встановити прапорець ignore_dangling_symlinks=True, щоб приховати цю помилку.

Якщо ignore не None, то це має бути функція, яка приймає як аргументи ім'я директорії, в якій зараз copytree(), і список вмісту, що повертається os.listdir(). Т.к. copytree() викликається рекурсивно, ignore викликається 1 раз для кожної піддиректорії. Вона повинна повертати список об'єктів щодо поточного імені директорії (тобто підмножина елементів у другому аргументі). Ці об'єкти не будуть скопійовані.

shutil.ignore_patterns(*patterns)

- функція, яка створює функцію, яка може бути використана як ignore для copytree(), ігноруючи файли та директорії, які відповідаютьglob-стилі шаблонів.

Наприклад:

```
copytree(source, destination,
ignore=ignore_patterns('*.pyc', 'tmp*'))
```

```
# Копіює всі файли, крім тих, що закінчуються на .рус
або починаються з tmp
```

shutil.rmtree(path, ignore_errors=False, onerror=None)

- видаляє поточну директорію та всі піддиректорії; path повинен вказувати на директорію, а не символічне посилання.

Якщо ignore_errors=True, то помилки, що виникають в результаті невдалого видалення, будуть проігноровані. Якщо False (за замовчуванням), ці помилки будуть передаватися обробнику onerror, або, якщо його немає, то виняток.

На ОС, які підтримують функції на основі файлових дескрипторів, за замовчуванням використовується версія rmtree(), не вразлива до атак символічних посилань.

На інших платформах це не так: за вибраного часу та обставин "хакер" може, маніпулюючи посиланнями, видалити файли, які недоступні йому в інших обставинах.

Щоб перевірити, чи вразлива система до подібних атак, можна використовувати атрибут rmtree.avoids_symlink_attacks.

Якщо заданий haerror, це має бути функція з 3 параметрами: function, path, excinfo.

Перший параметр, function, це функція, яка створила виняток; вона залежить від платформи та інтерпретатора. Другий параметр, path, це шлях, що передається функції. Третій параметр excinfo - це інформація про виключення, що повертається sys.exc_info(). Винятки, спричинені нанебезпеку, не обробляються.

shutil.move(src, dst, copy_function = copy2)

- рекурсивно переміщає файл або директорію (src) до іншого місця (dst), і повертає місце призначення.

Якщо dst – існуюча директорія, то src переміщається всередину директорії. Якщо dst існує, але з директорія, воно може бути перезаписано.

shutil.disk_usage(Path)

- повертає статистику використання дискового простору як намідтуплю з атрибутами total, used і free, в байтах.

shutil.chown(path, user=None, group=None)

- змінює власника та/або групу у файлу чи директорії.

shutil.which(cmd, mode=os.F_OK | os.X_OK, path=None)

- Повертає шлях до виконуваного файлу по заданій команді.

Якщо немає відповідності з жодним файлом, то None. mode це права доступу, потрібні від файлу, за замовчуванням шукає лише виконувані.

Архівація

Високорівневі функції для створення та читання архівованих та стислих файлів. Засновані на функціях із модулів zipfile та tarfile.

shutil.make_archive(base_name, format[, root_dir[, base_dir[, verbose[, dry_run[, owner[, group[, logger]]]]]]])

- Створює архів і повертає його ім'я.

`base_name`це ім'я файлу для створення, включаючи шлях, але не включаючи розширення (не потрібно писати ".zip" і т.д.).

`format`- Формат архіву.

`root_dir`- Директорія (щодо поточної), яку ми архівуємо.

`base_dir`- директорія, в яку архівуватиметься (тобто всі файли в архіві будуть у цій папці).

Якщо dry_run=True, архів не буде створений, але операції, які мали бути виконані, запишуться в logger.

`owner`та group використовуються при створенні tar-архіву.

shutil.get_archive_formats()

- Список доступних форматів для архівування.

```
>>>
>>> shutil.get_archive_formats()
[('bztar', "bzip2'ed tar-file"),
('gztar', "gzip'ed tar-file"),
('tar', 'uncompressed tar file'),
('xztar', "xz'ed tar-file"),
('zip', 'ZIP file')]
```

shutil.unpack_archive(filename[, extract_dir[, format]])

- Розпаковує архів. filename - повний шлях до архіву.

`extract_dir`- те, куди витягуватиметься вміст (за замовчуванням у поточну).

`format`- формат архіву (за замовчуванням намагається вгадати розширення файла).

shutil.get_unpack_formats() – список доступних форматів для розархівування.

Запит розміру терміналу виведення

shutil.get_terminal_size(fallback = (columns, lines))

- Повертає розмір вікна терміналу.

`fallback`повернеться, якщо не вдалося дізнатися про розмір терміналу (термінал не підтримує такі запити, або програма працює без терміналу). За замовчуванням (80, 24).

```
>>>
>>> shutil.get_terminal_size()
```

```
os.terminal_size(columns=102, lines=29)
>>> shutil.get_terminal_size() # Зменшили вікно
os.terminal_size(columns=67, lines=17)
```

XII. Приклади використання модуля shutil.

Копіювання файлу

Функція shutil.copyfile() копіює вміст джерела в місце призначення та викликає виключення IOError, якщо у нього немає дозволу на запис до файлу призначення.

```
>>> import shutil, os
>>> from glob import glob
# створимо тимчасову директорію
>>> os.mkdir('example')
# створимо тестовий файл
>>> open('example/test_file.txt', 'w').close()
# Копіювання
>>> shutil.copyfile('example/test_file.txt',
'example/test_file.txt.copy')
# 'example/test-file.txt.copy'

# дивимося результат
>>> pprint.pprint(glob('example/*'))
# ['example/test_file.txt.copy',
'example/test_file.txt']

# видаляємо
>>> shutil.rmtree('example')
```

Функція[shutil.copy()](#) інтерпретує ім'я вихідного файлу як інструмент командного рядка з Unix cp. Якщо шлях призначення вказаний як каталог, а не файл, то в каталозі створюється новий файл із використанням його базового імені.

```
>>> import shutil, os
>>> from glob import glob
# створимо тестовий файл
>>> open('shutil_copy.txt', 'w').close()
# створимо тимчасову директорію
>>> os.mkdir('example')
>>> glob('example/*')
# []

# Копіюємо
>>> shutil.copy('shutil_copy.txt', 'example')
```

```
# 'example/shutil_copy.txt'

# дивимося результат
>>> glob('example/*')
# ['example/shutil_copy.txt']

# видаляємо
>>> shutil.rmtree('example')
```

Рекурсивне копіювання каталогу

Функція[shutil.copytree()](shutil.copytree()) рекурсивно копіює весь каталог.

```
# для того, щоб протестувати функцію copytree()
# створимо ієрархію каталогів
import pathlib, itertools, glob, shutil
path = 'test/'
tmp = {'script':'.py', 'text':'.txt'}
for d, ext in tmp.items():
    comb = itertools.combinations([d, 1, 0], r=2)
    for a, b in comb:
        name = f'{a}{b}{ext}'
        pathlib.Path(path, d).mkdir(parents=True,
exist_ok=True)
        pathlib.Path(path, d,
name).touch(exist_ok=True)

f = glob.glob('test/**/*', recursive=True)
print(f)
# ['test/text', 'test/script', 'test/text/10.txt',
# 'test/text/text1.txt', 'test/text/text0.txt',
# 'test/script/10.py', 'test/script/script0.py',
# 'test/script/script1.py']

# Копіюємо
shutil.copytree('test', 'test_cp')
# дивимося результат
f_cp = glob.glob('test_cp/**/*', recursive=True)
print(f_cp)

# ['test/text', 'test/script', 'test/text/10.txt',
# 'test/text/text1.txt', 'test/text/text0.txt',
# 'test/script/10.py', 'test/script/script0.py',
# 'test/script/script1.py']

# видаляємо
shutil.rmtree('test_cp')
```

```
shutil.rmtree('test')
```

Вибіркове рекурсивне копіювання файлів каталогу

Приклад, який використовує помічникshutil.ignore_patterns().

Тут копіюється все, крім файлів .pyc і файлів або каталогів, ім'я яких починається з tmp.

```
від shutil import copytree, ignore_patterns
copytree(source, destination,
ignore=ignore_patterns('*.pyc', 'tmp*'))
```

Інший приклад, який використовує аргумент ignore для додавання виклику логування копіювання файлів:

```
від shutil import copytree
import logging

def _logpath(path, names):
    logging.info('Working in %s', path)
    return [] # nothing will be ignored

copytree(source, destination, ignore=_logpath)
```

Рекурсивне видалення каталогу

Робота функціїshutil.rmtree(),яка виконує рекурсивне видалення каталогу демонструвалася вище. Розберемо ситуації складніше.

У цьому прикладі показано, як видалити дерево каталогів у Windows, де для деяких файлів встановлений біт тільки для читання.

Він використовує зворотний виклик haerror, щоб очистити біт readonly і повторити спробу видалення.

```
import os, stat
import shutil

def remove_readonly(func, path, _):
    "Clear the readonly bit and reattempt the removal"
    os.chmod(path, stat.S_IWRITE)
    func(path)

shutil.rmtree(directory, onerror=remove_readonly)
```

У функціїshutil.rmtree(),так само можна використовувати помічникshutil.ignore_patterns()для вибіркового рекурсивного видалення файлів, як це робиться у вибірковому рекурсивному копіюванні файлів.

Приклад реалізації функції shutil.copytree()

Цей приклад є реалізацією функціїshutil.copytree()з опущеним рядком документації. Він демонструє багато інших функцій, що надаються цим модулем.

```
def copytree (src, dst, symlinks = False):
    names = os.listdir(src)
    os.makedirs(dst)
    errors = []
    for name in names:
        srcname = os.path.join(src, name)
        dstname = os.path.join(dst, name)
        try:
            if symlinks and os.path.islink(srcname):
                linkto = os.readlink(srcname)
                os.symlink(linkto, dstname)
            elif os.path.isdir(srcname):
                copytree(srcname, dstname, symlinks)
            else:
                copy2(srcname, dstname)
            # XXX What about devices, sockets etc.?
        except OSError as why:
            errors.append((srcname, dstname, str(why)))
        # catch the Error from the recursive copytree so that we
can
        # continue with other files
        except Error as err:
            errors.extend(err.args[0])
    try:
        copystat(src, dst)
    except OSError as why:
        # can't copy file access times on Windows
        if why.winerror is None:
            errors.extend((src, dst, str(why)))
    if errors:
        raise Error(errors)
```

Архівування каталогів

У цьому прикладі створимо архів tar-файлів з використанням gzip, що містить усі файли, знайдені в каталозі .ssh користувача:

```
від shutil import make_archive
import os
```

```
archive_name = os.path.expanduser(os.path.join('~',
'myarchive'))

root_dir = os.path.expanduser(os.path.join('~',
                                           '.ssh'))
make_archive(archive_name, 'gztar', root_dir)

'/Users/docs-python/myarchive.tar.gz'
```

Отриманий архів містить:

```
~$ tar -tzvf /home/docs-python/myarchive.tar.gz

drwx------ docs-python/docs-python 0 2019-12-17 16:57 ./
-rw------- docs-python/docs-python 1554 2019-12-17 16:53 ./known_hosts
-rw------- docs-python/docs-python 1554 2019-12-17 14:06 ./known_hosts.old
-rw------- docs-python/docs-python 1679 2019-09-16 12:02 ./id_rsa
-rw-r--r-- docs-python/docs-python 399 2019-09-16 12:02 ./id_rsa.pub
```

XIII. Модуль pathlib

Python 3 включає модуль pathlib для маніпуляції шляхами файлових систем незалежно від операційної системи.

pathlibсхожий на модуль os.path, але pathlib пропонує більш розвинений та зручний інтерфейс порівняно з os.path.

Зазвичай файли на комп'ютері ідентифікують за допомогою ієрархічних шляхів.

Наприклад, можна ідентифікувати файл wave.txt на комп'ютері за допомогою цього шляху: /Users/sammy/ocean/wave.txt.

Операційні системи представляють шляхи дещо по-різному. Windows може представляти шлях до файлу wave.txt як C:\Users\sammy\ocean\wave.txt.

Модуль pathlib може бути корисним, якщо у програмі Python ви створюєте або переміщаєте файли у файловій системі, вказуючи всі файли у файловій системі, що збігаються з даним розширенням або шаблоном, або створюєте шляхи файлу, що відповідають файловій системі на основі наборів неформатованих рядків.

Хоча ви можете використовувати інші інструменти, наприклад модуль os.path, для виконання більшої частини цих завдань, але модуль pathlib дозволяє виконувати ці операції з більшим ступенем читання та мінімальною кількістю кодів.

Розглянемо деякі способи використання модуля pathlib для представлення та маніпуляції шляхами файлових систем.

Модуль pathlib надає кілька класів, але одним із найважливіших є клас Path.

Примірники класу Path представляють шлях до файлу або каталогу файлової системи вашого комп'ютера.

Наприклад, наступний код отримує екземпляр Path, який представляє частину шляху до файлу wave.txt:

```
від pathlib import Path

wave = Path("ocean", "wave.txt")
print(wave)
```

Якщо запустити цей код, результат буде виглядати так:

```
Output
ocean/wave.txt
```

`від pathlib import Path`робить клас Path доступним для нашої програми.

Потім Path("ocean", "wave.txt") отримує новий екземпляр Path.

З висновку результату видно, що Python «додав» відповідний роздільник оперативної системи між двома заданими нами компонентами шляху "ocean" і "wave.txt".

Примітка.Залежно від операційної системи, висновок може трохи відрізнятися від прикладів, наведених у цьому посібнику. У Windows, наприклад, висновок цього прикладу може виглядати як ocean\wave.txt.

У прикладі об'єкт Path, наданий змінною wave, містить відносний шлях.

Можна використовувати Path.home() для отримання абсолютного шляху до домашнього каталогу поточного користувача:

```
home = Path.home()
wave_absolute = Path(home, "ocean", "wave.txt")
print(home)
print(wave_absolute)
```

Якщо запустити цей код, результат буде виглядати приблизно так:

```
Output
/Users/sammy
/Users/sammy/ocean/wave.txt
```

Примітка.Як згадувалося раніше, висновок залежатиме від операційної системи

`Path.home()`повертає екземпляр Path з абсолютним шляхом до домашнього каталогу поточного користувача.

Потім ми передамо цей екземпляр Path і рядки "ocean" і "wave.txt" до іншого конструктора Path, щоб створити абсолютний шлях до файлу wave.txt.

Висновок показує, що перший рядок – це домашній каталог, а другий рядок – домашній каталог плюс ocean/wave.txt.

Цей приклад також ілюструє важливу функцію класу Path: конструктор Path приймає обидві рядки і об'єкти Path, що раніше існували.

Давайте детальніше розглянемо підтримку рядків та об'єктів Path у конструкторі Path:

```
shark = Path(Path.home(), "ocean", "animals",
Path("fish", "shark.txt"))

print(shark)
```

Якщо запустити цей код Python, результат буде виглядати так:

```
Output
/Users/sammy/ocean/animals/fish/shark.txt
```

`shark`— це Path до файлу, який ми створили за допомогою об'єктів Path (Path.home() та Path("fish", "shark.txt") та рядків "ocean" та "animals").

Конструктор Path інтелектуально обробляє обидва типи об'єктів та акуратно з'єднує їх за допомогою відповідного роздільника операційної системи, в даному випадку /.

Доступ до атрибутів файлу

Розглянемо, як можна використовувати екземпляри Path для доступу до інформації про файл.

Ми можемо використовувати атрибути name і suffix для доступу до імен та розширень файлів:

```
wave = Path("ocean", "wave.txt")
print(wave)
print(wave.name)
print(wave.suffix)
```

Запустивши цей код і отримаємо висновок, аналогічний до наступного:

```
Output
/Users/sammy/ocean/wave.txt
wave.txt
.txt
```

Цей висновок показує, що ім'я файлу в кінці нашого шляху wave.txt, а розширення файлу .txt.

Примірники Path також пропонують функцію with_name, що дозволяє без перешкод створювати новий об'єкт Path з іншим ім'ям:

```
wave = Path("ocean", "wave.txt")
tides = wave.with_name("tides.txt")
print(wave)
print(tides)
```

Якщо запустити його, результат буде виглядати так:

```
ocean/wave.txt
ocean/tides.txt
```

Код спершу створює екземпляр Path, який вказує на файл з ім'ям wave.txt.

Потім викликається метод with_name у wave, щоб повернути другий екземпляр Path, що вказує на новий файл з ім'ям tides.txt.

Частина каталогу ocean/ залишається незміненою та залишає фінальний шлях у вигляді ocean/tides.txt

Доступ до попередніх об'єктів

Іноді корисно отримати доступ до каталогів, які містять певний шлях. Давайте розглянемо приклад:

```
shark = Path("ocean", "animals", "fish", "shark.txt")
print(shark)
print(shark.parent)
```

Якщо запустити цей код, результат буде виглядати так:

```
Output
ocean/animals/fish/shark.txt
ocean/animals/fish
```

Атрибут parent в екземплярі Path повертає найближчого попередника шляху файлу. У цьому випадку він повертає каталог із файлом shark.txt: ocean/animals/fish.

Можемо отримувати доступ до атрибуту parent кілька разів у команді, щоб пройти вгору кореневим деревом даного файлу:

```
shark = Path("ocean", "animals", "fish", "shark.txt")
print(shark)
print(shark.parent.parent)
```

Якщо виконати цей код, побачимо наступні:

```
Output
ocean/animals/fish/shark.txt
ocean/animals
```

Висновок буде схожий на попередній висновок, але тепер ми перейшли на рівень вище, отримавши доступ до .parent вдруге. Два каталоги від shark.txt – це каталог ocean/animals.

Використання шаблону пошуку для списку файлів

Також можна використовувати клас Path для списку файлів за допомогою glob.

Припустимо, у нас є структура каталогу, яка виглядає так:

```
└── ocean
    ├── animals
    │   └── fish
    │   └── shark.txt
    ├── tides.txt
    └── wave.txt
```

Каталог ocean містить файли tides.txt та wave.txt. У нас є файл з ім'ям shark.txt, вкладений у каталог ocean, каталог animals та каталог fish: ocean/animals/fish.

Щоб перерахувати всі файли .txt у каталозі ocean, можна використовувати:

```
for txt_path in Path("ocean").glob("*.txt"):
    print(txt_path)
```

Цей код зробить такий висновок:

```
Output
ocean/wave.txt
ocean/tides.txt
```

Шаблон пошуку "*.txt" знаходить усі файли, що закінчуються на .txt. Оскільки приклад коду виконує цей пошук у каталозі ocean, він повертає два файли .txt у каталозі ocean: wave.txt та tides.txt.

Також можна використовувати метод glob рекурсивно.

Щоб перерахувати всі файли .txt у каталозі ocean і всі його підкаталоги, запишемо:

```
for txt_path in Path("ocean").glob("**/*.txt"):
    print(txt_path)
```

Якщо запустити цей код, результат буде виглядати так:

```
Output
ocean/wave.txt
ocean/tides.txt
ocean/animals/fish/shark.txt
```

Частина ** шаблону пошуку буде відповідати цьому каталогу та всім каталогам під ним рекурсивно.

Тому у висновку у нас будуть не тільки файли wave.txt та tides.txt, але також ми отримаємо файл shark.txt, вкладений у ocean/animals/fish.

Обчислення відносних шляхів

Можна використовувати метод Path.relative_to для обчислення шляхів, які стосуються один одного. Метод relative_to корисний, якщо, наприклад, ви хочете отримати частину довгого шляху файлу.

Розгляньте наступний код:

```
shark = Path("ocean", "animals", "fish", "shark.txt")
below_ocean = shark.relative_to(Path("ocean"))
below_animals = shark.relative_to(Path("ocean",
"animals"))

print(shark)
print(below_ocean)
print(below_animals)
```

Якщо запустити його, результат буде виглядати так:

```
Output
ocean/animals/fish/shark.txt
animals/fish/shark.txt
fish/shark.txt
```

Метод relative_to повертає новий об'єкт Path, що належить до цього аргументу. У нашому прикладі ми обчислимо Path до shark.txt, що відноситься до каталогу ocean, а потім стосується обох каталогів ocean і animals.

Якщо relative_to не зможе вирахувати відповідь, оскільки ми даємо йому не пов'язаний шлях, він видасть ValueError:

```
shark = Path("ocean", "animals", "fish", "shark.txt")
shark.relative_to(Path("unrelated", "path"))
```

Ми отримаємо виняток ValueError, що виник з цього коду, який виглядатиме так:

```
Output
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/local/lib/Python3.8/pathlib.py", line 899, in
relative_to
    raise ValueError("{!r} does not start with {!r}"
ValueError: 'ocean/animals/fish/shark.txt' does not start with
'unrelated/path'
```

`unrelated/path`не є частиною ocean/animals/fish/shark.txt, тому Python не зможе обчислити відносний шлях.

Модуль pathlib представляє додаткові класи та утиліти, Можна скористатися посиланням надокументацію модуля pathlibдля отримання додаткової інформації.

XIV. Модуль glob

Модуль glob знаходить всі шляхи, що збігаються із заданим шаблоном відповідно до правил, що використовуються оболонкою Unix.

Обробляються символи "*" (довільна кількість символів), "?" (один символ) та діапазони символів за допомогою [].

Для використання тильди "~" та змінних оточення необхідно використовуватиos.path.expanduser() та os.path.expandvars().

Для пошуку спецсимволів укладайте їх у квадратні дужки. Наприклад, [?] відповідає символу "?".

glob.glob(pathname)

- Повернення списку (можливо, порожній) шляхів, що відповідають шаблону pathname. Шлях може бути абсолютним (наприклад, /usr/src/Python-1.5/Makefile) або відносний (як ../../Tools/*/*.gif).

glob.iglob(pathname)

- Повертає ітератор, що дає ті ж значення, що й glob.glob.

glob.escape(pathname)

- екранує всі спеціальні символи для glob ("?", "*" та "["). Спеціальні символи в імені диска не екрануються (оскільки вони там не враховуються), тобто у Windows escape("//?/c:/Quo vadis?.txt") повертає "//?/c:/Quo vadis[?].txt".

Розглянемо, наприклад, каталог, що містить лише такі файли: 1.gif, 2.txt та card.gif.

glob.glob() поверне наступні результати. (Зверніть увагу, що будь-які провідні компоненти шляху зберігаються):

```
>>> import glob
>>> glob.glob('./[0-9].*')
['./1.gif', './2.txt']
>>> glob.glob('*.gif')
['1.gif', 'card.gif']
>>> glob.glob('?.gif')
['1.gif']
```

Якщо каталог містить файли, що починаються з ".", вони не включатимуться за замовчуванням. Розглянемо, наприклад, каталог, що містить card.gif та .card.gif:

```
>>>
```

```
>>> import glob
>>> glob.glob('*.gif')
['card.gif']
>>> glob.glob('.c*')
['.card.gif']
```

XV. Модуль os.path

os.pathє вкладеним модулем модуль os, і реалізує деякі корисні функції для роботи з шляхами.

os.path.abspath(path) – повертає нормалізований абсолютний шлях.

os.path.basename(path) – базове ім'я шляху (еквівалентно os.path.split(path) [1]).

os.path.commonprefix(list) - повертає найдовший префікс усіх шляхів у списку.

os.path.dirname(path) – повертає ім'я директорії шляху path.

os.path.exists(path) – повертає True, якщо path вказує на існуючий шлях або дескриптор відкритого файлу.

os.path.expanduser(path) - замінює ~ або ~user на домашню директорію користувача.

os.path.expandvars(path) - повертає аргумент із підставленими змінними оточення ($name або ${name} замінюються змінною оточення name). Неіснуючі імена не замінює. Windows також замінює %name%.

os.path.getatime(Path) - час останнього доступу до файлу, в секундах.

os.path.getmtime(Path) - час останньої зміни файлу, в секундах.

os.path.getctime(path) – час створення файлу (Windows), час останньої зміни файлу (Unix).

os.path.getsize(Path) - розмір файлу в байтах.

os.path.isabs(path) – чи є шлях абсолютним.

os.path.isfile(path) – чи є шлях файлом.

os.path.isdir(path) – чи є шлях директорією.

os.path.islink(path) – чи є шлях символічним посиланням.

os.path.ismount(path) – чи є шлях точкою монтування.

os.path.join(path1[, path2[, ...]]) - з'єднує шляхи з урахуванням особливостей операційної системи.

os.path.normcase(path) - нормалізує регістр шляху (на файлових системах, які не враховують регістр, наводить шлях до нижнього регістру).

os.path.normpath(path) – нормалізує шлях, прибираючи надлишкові роздільники та посилання на попередні директорії. На Windows перетворює прямі сліші на зворотні.

os.path.realpath(path) - повертає канонічний шлях, забираючи всі символічні посилання (якщо вони підтримуються).

os.path.relpath(path, start=None) - обчислює шлях щодо директорії start (за умовчанням - щодо поточної директорії).

os.path.samefile(path1, path2) - чи вказують path1 і path2 на той самий файл або директорію.

os.path.sameopenfile(fp1, fp2) - чи вказують дескриптори fp1 і fp2 на той самий відкритий файл.

os.path.split(path) – розбиває шлях на кортеж (голова, хвіст), де хвіст – останній компонент шляху, а голова – все інше. Хвіст ніколи не починається зі сліша (якщо шлях закінчується слішем, то хвіст порожній). Якщо слішів у дорозі немає, то порожньою буде голова.

os.path.splitdrive(path) – розбиває шлях на пару (привід, хвіст).

os.path.splitext(Path) - розбиває шлях на пару (root, ext), де ext починається з точки і містить не більше однієї точки.

os.path.supports_unicode_filenames- Чи підтримує файлова система Unicode.

XVI. Модуль sys

Модуль sys забезпечує доступ до деяких змінних та функцій, що взаємодіють з інтерпретатором python.

sys.argv- Перелік аргументів командного рядка, що передаються сценарієм Python. sys.argv[0] є ім'ям скрипта (порожнім рядком в інтерактивній оболонці).

Прикладниквивести усі аргументи командного рядка: for param in sys.argv: print (param)

sys.byteorder- Порядок байтів. Матиме значення 'big' при порядку проходження бітів від старшого до молодшого, і 'little', якщо навпаки (молодший байт перший).

sys.builtin_module_names- Кортеж рядків, що містить імена всіх доступних модулів.

sys.call_tracing(функція, аргументи) - викликає функцію з аргументами та включеним трасуванням, у той час як трасування включене.

sys.copyright- Рядок, що містить авторські права, що відносяться до інтерпретатора Python.

sys._clear_type_cache() – очищає внутрішній кеш типу.

sys._current_frames() – повертає словник-відображення ідентифікатора для кожного потоку у верхньому кадрі стека в даний час у цьому потоці в момент виклику функції.

sys.dllhandle- ціле число, що визначає дескриптор DLL Python (Windows).

sys.exc_info() - повертає кортеж із трьох значень, які дають інформацію про винятки, що опрацьовуються на даний момент.

sys.exec_prefix- каталог встановлення Python.

sys.executable- Шлях до інтерпретатора Python.

sys.exit([arg]) - вихід із Python. Порушує виняток SystemExit, який може бути перехоплений.

sys.flags- Прапори командного рядка. Атрибути лише для читання.

sys.float_info- інформація про тип даних float.

sys.float_repr_style- інформація про застосування вбудованої функції repr() типу float.

sys.getdefaultencoding() - повертає використовуване кодування.

sys.getdlopenflags() – значення прапорів для викликів dlopen().

sys.getfilesystemencoding() – повертає кодування файлової системи.

sys.getrefcount(object) – повертає кількість посилань на об'єкт. Аргумент функції getrefcount - ще одне посилання на об'єкт.

sys.getrecursionlimit() - Повертає ліміт рекурсії.

sys.getsizeof(object [, default]) – повертає розмір об'єкта (в байтах).

sys.getswitchinterval() – інтервал перемикання потоків.

sys.getwindowsversion() – повертає кортеж, що описує версію Windows.

sys.hash_info- інформація про параметри хешування.

sys.hexversion- Версія python як шістнадцяткове число (для 3.2.2 final це буде 30202f0).

sys.implementation- об'єкт, що містить інформацію про запущеного python інтерпретатора.

sys.int_info- інформація про тип int.

sys.intern(Рядок) - повертає інтернований рядок.

sys.last_type, sys.last_value, sys.last_traceback- інформація про оброблені винятки. За змістом схоже sys.exc_info().

sys.maxsize- максимальне значення числа типу Py_ssize_t (231 на 32-бітових та 263 на 64-бітних платформах).

sys.maxunicode- Максимальна кількість біт для зберігання символу Unicode.

sys.modules- словник імен завантажених модулів. Змінюємо, тому можна потішитися :)

sys.path- Перелік шляхів пошуку модулів.

sys.path_importer_cache- словник-кеш для пошуку об'єктів.

sys.platform- інформація про операційну систему.

Linux (2.x і 3.x)	'linux'
Windows	'win32'
Windows/Cygwin	'cygwin'
Mac OS X	'darwin'
OS/2	'os2'
OS/2 EMX	'os2

	emx'

sys.prefix- папка встановлення інтерпретатора python.

sys.ps1, sys.ps2- первинне та вторинне запрошення інтерпретатора (визначено лише якщо інтерпретатор перебуває в інтерактивному режимі). За промовчанням sys.ps1 == ">>> ", а sys.ps2 == "... ".

sys.dont_write_bytecode- якщо true, python не писатиме .pyc файли.

sys.setdlopenflags(flags) – встановити значення прапорів для викликів dlopen().

sys.setrecursionlimit(Між) - встановити максимальну глибину рекурсії.

sys.setswitchinterval(інтервал) - Встановити інтервал перемикання потоків.

sys.settrace(tracefunc) - встановити "слід" функції.

sys.stdin- Стандартне введення.

sys.stdout- Стандартний висновок.

sys.stderr- Стандартний потік помилок.

sys.__stdin__, sys.__stdout__, sys.__stderr__- вихідні значення потоків введення, виведення та помилок.

sys.tracebacklimit- Максимальна кількість рівнів відстеження.

sys.version- Версія python.

sys.api_version- Версія C API.

sys.version_info- Кортеж, який містить п'ять компонентів номера версії.

sys.warnoptions- Реалізація попереджень.

sys.winver- Номер версії python, який використовується для формування реєстру Windows.

XVII. Модуль itertools

Модуль itertools – збірка корисних ітераторів.

`itertools.count`(start = 0, step = 1)

- нескінченна арифметична прогресія з першим членом start та кроком step.

`itertools.cycle`(iterable)

- Повертає по одному значенню з послідовності, повтореної нескінченне число разів.

`itertools.repeat`(Elem, n = Inf)

- Повторює elem n разів.

`itertools.accumulate`(iterable) – акумулює суми.

```
accumulate([1,2,3,4,5]) --> 1 3 6 10 15
```

`itertools.chain`(*iterables)

- Повертає по одному елементу з першого ітератора, потім з другого, доки ітератори не закінчаться.

itertools.combinations(iterable, [r])

- комбінації довжиною r з iterable без повторюваних елементів.

```
combinations('ABCD', 2) --> AB AC AD BC BD CD
```

itertools.combinations_with_replacement(iterable, r)

- комбінації довжиною r з iterable з елементами, що повторюються.

```
combinations_with_replacement('ABCD', 2) --> AA AB AC
AD BB BC BD CC CD DD
```

itertools.compress(data, selectors) - (d[0] if s[0]), (d[1] if s[1]), ...

```
compress('ABCDEF', [1,0,1,0,1,1]) --> ACEF
```

itertools.dropwhile(func, iterable)

- елементи iterable, починаючи з першого, для якого func повернула брехню.

```
dropwhile(lambda x: x < 5, [1,4,6,4,1]) --> 6 4 1
```

itertools.filterfalse(func, iterable)

- всі елементи, для яких func повертає брехню.

itertools.groupby(iterable, key=None)

- Групує елементи за значенням. Значення виходить застосуванням функції key до елемента (якщо аргумент key не вказано, значенням є сам елемент).

```
>>>
>>> від itertools import groupby
>>> things = [("animal", "bear"), ("animal", "duck"),
("plant", "cactus"),

... ("vehicle", "speed boat"), ("vehicle", "school
bus")]

>>> для key, group in groupby(things, lambda x: x[0]):
... for thing in group:
... print("A %s is a %s." % (thing[1], key))
... print()
Bear is a animal.
Duck is a animal.

A cactus is a plant.
```

```
A speed boat is a vehicle.
A school bus is a vehicle.
```

itertools.islice(iterable[, start], stop[, step])

- ітератор, що складається із зрізу.

-

itertools.permutations(iterable, r=None)

- перестановки завдовжки r з iterable.

itertools.product(*iterables, repeat=1)

- аналог вкладених циклів.

```
product('ABCD', 'xy') --> Ax Ay Bx By Cx Cy Dx Dy
```

itertools.starmap(function, iterable)

- застосовує функцію кожного елемента послідовності (кожен елемент розпаковується).

```
starmap(pow, [(2,5), (3,2), (10,3)]) --> 32 9 1000
```

itertools.takewhile(func, iterable)

- Елементи до тих пір, поки func повертає істину.

```
takewhile(lambda x: x<5, [1,4,6,4,1]) --> 1 4
```

itertools.tee(iterable, n=2)

- Кортеж з n ітераторів.

itertools.zip_longest(*iterables, fillvalue=None)

- як вбудована функція zip, але бере найдовший ітератор, а короткі доповнює fillvalue.

```
zip_longest('ABCD', 'xy', fillvalue='-') --> Ax By C-D-
```

XVIII. Модуль locale

При форматуванні чисел Python за замовчуванням використовує англосаксонську систему, при якій розряди цілого числа відокремлюються один від одного комами, а частина від цілої відокремлюється точкою. У континентальній Європі, наприклад, використовується інша система, за якої розряди розділяються точкою, а дробова і ціла частина – комою (https://metanit.com/python/tutorial/6.3.php):

```
англосаксонська система
1,234.567
# європейська система
```

```
1.234,567
```

Для вирішення проблеми форматування під певну локаль Python є вбудований модуль locale.

Для з'ясування яка локаль встановлена можна виконати:

```
importlocallocale.getlocale()
```

Наприклад, висновок:

```
('Russian_Russia', '1251')
```

Для встановлення локалі в модулі locale визначено функцію setlocale(). Вона приймає два параметри:

```
setlocale(category, locale)
```

Перший параметр вказує на категорію, до якої застосовується функція - до чисел, валют або числах, і валют. Як значення для параметра можна передавати одну з наступних констант:

- locale.LC_ALL: застосовує локалізацію до всіх категорій – до форматування чисел, валют, дат тощо.
- locale.LC_NUMERIC: застосовує локалізацію до чисел
- locale.LC_MONETARY: застосовує локалізацію до валют
- locale.LC_TIME: застосовує локалізацію до дат та часу
- locale.LC_CTYPE: застосовує локалізацію під час перекладу символів у верхній або нижній регістр
- locale.LC_COLLIATE: застосовує локаль для порівняння рядків

Другий параметр функції setlocale вказує на локаль, яку потрібно використовувати.

На Windows можна використовувати код країни за ISO із двох символів:

для США - "us",

для Німеччини - "de",

для Росії – "ru".

Але на MacOS необхідно вказувати код мови та код країни, наприклад, для англійської в США - "en_US", для німецької в Німеччині - "de_DE", для російської в Росії - "ru_RU". За промовчанням фактично використовується локаль "en_US".

Безпосередньо для форматування чисел та валют модуль locale надає дві функції:

- `currency(num)` −форматує валюту
- `format(str, num)` −підставляє число num замість плейсхолдера в рядок str

Застосовуються такі плейсхолдери:

- `d`: для цілих чисел
- `f`: для чисел з плаваючою точкою
- `e`: для експоненційного запису чисел

Перед кожним плейсхолдером ставиться знак відсотка %, наприклад:

```
"%d"
```

При виведенні дробових чисел перед плейсхолдером після точки можна вказати скільки знаків у дробовій частині повинно відображатися:

```
%.2f # два знаки в дрібній частині
```

Застосуємо локалізацію чисел та валют у німецькій локалі:

```
import locale

locale.setlocale(locale.LC_ALL, "de") # для Windows
# locale.setlocale(locale.LC_ALL, "de_DE")# для MacOS

number = 12345.6789
formatted = locale.format("%f", number)
print(formatted) # 12345,678900

formatted = locale.format("%.2f", number)
print(formatted) # 12345,68

formatted = locale.format("%d", number)
print(formatted) # 12345

formatted = locale.format("%e", number)
print(formatted) # 1,234568e+04

гроші = 234.678
formatted = locale.currency(money)
print(formatted) # 234,68 €
```

Якщо замість конкретного коду як другий параметр функції setlocale() передати порожній рядок, то Python буде використовувати локаль, що застосовується на поточній робочій машині.

За допомогою функції getlocale() можна дізнатися про поточну локаль:

```
import locale

locale.setlocale(locale.LC_ALL, "")

number = 12345.6789
formatted = locale.format("%.02f", number)
print(formatted) # 12345,68
print(locale.getlocale()) # ('Russian_Russia', '1251')
```

XIX. Модуль datetime

Основний функціонал для роботи з датами та часом зосереджений у модулі datetime у вигляді наступних класів (за матеріалами https://metanit.com/python/):

Клас date

Для роботи з датами скористаємося класом date, визначеним у модулі datetime. Для створення об'єкта date ми можемо використовувати конструктор date, який послідовно приймає три параметри: рік, місяць та день:

```
date(year, month, day)
```

Наприклад, створимо якусь дату:

```
importdatetime

yesterday = datetime.date(2017,5, 2)
print(yesterday) # 2017-05-02
```

Якщо необхідно отримати поточну дату, можна скористатися методом today():

```
fromdatetime import date

today = date.today()
print(today) # 2017-05-03
print("{}.{}.{}".format(today.day, today.month,
today.year)) # 2.5.2017
```

За допомогою властивостей day, month, year можна отримати відповідно день, місяць та рік

Клас time

За роботу з часом відповідає клас часу. Використовуючи його конструктор, можна створити об'єкт часу:

```
time([hour] [, min] [, sec] [, microsec])
```

Конструктор послідовно приймає години, хвилини, секунди та мікросекунди. Усі параметри необов'язкові, і якщо ми якийсь параметр не передамо, то відповідне значення ініціалізуватиметься нулем.

```
from datetime import time

current_time = time()
print(current_time) # 00:00:00

current_time = time(16, 25)
print(current_time) # 16:25:00

current_time = time(16, 25, 45)
print(current_time) # 16:25:45
```

Клас datetime

Клас datetime з однойменного модуля поєднує можливості роботи з датою та часом. Для створення об'єкта datetime можна використовувати наступний конструктор:

```
datetime(year, month, day [, hour] [, min] [, sec] [,
microsec])
```

Перші три параметри, що становлять рік, місяць і день, є обов'язковими. Інші необов'язкові, і якщо ми не вкажемо для них значення, то за умовчанням вони ініціалізуються банкрутом.

```
fromdatetime import datetime
deadline = datetime(2017, 5, 10)
print(deadline) # 2017-05-10 00:00:00

deadline = datetime(2017, 5, 10, 4, 30)
print(deadline) # 2017-05-10 04:30:00
```

Для отримання поточної дати та часу можна викликати метод now():

```
від datetime import datetime

now = datetime.now()
print(now) # 2017-05-03 11:18:56.239443

print("{}.{}.{} {}:{}".format(now.day, now.month,
now.year, now.hour, now.minute)) # 3.5.2017 11:21

print(now.date())
print(now.time())
```

За допомогою властивостей day, month, year, hour, minute, second можна отримати окремі значення дати та часу. А через методи date() та time() можна отримати окремо дату та час відповідно.

Перетворення з рядка на дату

З функціональності класу datetime слід зазначити метод strptime(), який дозволяє розпарсувати рядок та перетворити його на дату. Цей метод приймає два параметри:

```
strptime(str, format)
```

Перший параметр str представляє рядкове визначення дати та часу, а другий параметр - формат, який визначає, як різні частини дати та часу розташовані у цьому рядку.

Для визначення формату можна використовувати такі коди:

- **%d**: день місяця у вигляді числа
- **%m**: порядковий номер місяця
- **%y**: рік у вигляді 2-х чисел
- **%Y**: рік у вигляді 4-х чисел

- **%H**: година у 24-х годинному форматі.
- **%M**: хвилина
- **%S**: секунда

Застосуємо різні формати:

```
fromdatetime import datetime
deadline = datetime.strptime("22/05/2017", "%d/%m/%Y")
print(deadline) # 2017-05-22 00:00:00

deadline = datetime.strptime("22/05/2017 12:30",
"%d/%m/%Y %H:%M")
print(deadline) # 2017-05-22 12:30:00

deadline = datetime.strptime("05-22-2017 12:30", "%m-
%d-%Y %H:%M")

print(deadline) # 2017-05-22 12:30:00
```

Операції з датами

Фотування дат та часу

Для форматування об'єктів date і time в обох класах передбачений метод strftime(format). Цей метод приймає лише один параметр, що вказує на формат, який потрібно перетворити дату або час.

Для визначення формату можна використовувати один із таких кодів форматування:

- %a: абревіатура дня тижня. Наприклад, Wed – від слова Wednesday (за замовчуванням використовуються англійські найменування)
- %A: день тижня повністю, наприклад, Wednesday
- %b: абревіатура назви місяця. Наприклад, Oct (скорочення від October)
- %B: назва місяця повністю, наприклад, October
- %d: день місяця, доповнений нулем, наприклад, 01
- %m: номер місяця, доповнений нулем, наприклад, 05
- %y: рік у вигляді 2-х чисел
- %Y: рік у вигляді 4-х чисел
- %H: година у 24-х годинному форматі, наприклад, 13
- %I: година у 12-ти годинному форматі, наприклад, 01
- %M: хвилина
- %S: секунда
- %f: мікросекунда
- %p: покажчик AM/PM
- %c: дата та час, відформатовані під поточну локаль
- %x: дата, відформатована під поточну локаль

- %X: час, форматований під поточну локаль

Використовуємо різні формати:

```
від datetime import datetime
now = datetime.now()
print(now.strftime("%Y-%m-%d")) # 2017-05-03
print(now.strftime("%d/%m/%Y")) # 03/05/2017
print(now.strftime("%d/%m/%y")) # 03/05/17
print(now.strftime("%d %B %Y (%A)")) # 03 May 2017
(Wednesday)

print(now.strftime("%d/%m/%y %I:%M")) 03/05/17 01:36
```

При виведенні назв місяців та днів тижня за умовчанням використовуються англійські найменування. Якщо ми хочемо використовувати поточну локаль, але ми можемо її попередньо встановити за допомогою модуля locale:

```
від datetime import datetime
importlocale
locale.setlocale(locale.LC_ALL, "ru")

now = datetime.now()
print(now.strftime("%d %B %Y (%A)"))
```

Результат виконання скрипту:

```
06 Березень 2018 (вівторок)
```

Складання та віднімання дат і часу

Нерідко при роботі з датами виникає необхідність додати до будь-якої дати певний проміжок часу або, навпаки, відняти певний період. І спеціально для таких операцій в модулі datetime визначено клас timedelta. Фактично цей клас визначає певний період.

Для визначення проміжку часу можна використовувати конструктор timedelta:

```
timedelta([days] [, seconds] [, microseconds] [,
milliseconds] [, minutes] [, hours] [, weeks])
```

У конструктор послідовно передаються дні, секунди, мікросекунди, мілісекунди, хвилини, години та тижні.

Визначимо кілька періодів:

```
fromdatetime import timedelta

three_hours = timedelta(hours=3)
print(three_hours) # 3:00:00
three_hours_thirty_minutes = timedelta(hours=3,
minutes=30) # 3:30:00
```

```
two_days = timedelta(2) # 2 days, 0:00:00

two_days_three_hours_thirty_minutes = timedelta(days=2,
hours=3, minutes=30) # 2 days, 3:30:00
```

Використовуючи timedelta, можна складати або віднімати дати. Наприклад, отримаємо дату, яка буде за два дні:

```
від datetime import timedelta, datetime

now = datetime.now()
print(now)
two_days = timedelta(2)
in_two_days = now + two_days
print(in_two_days)
```

Висновок скрипту:

```
2018-03-06 12:20:48.524307
2018-03-08 12:20:48.524307
```

Або дізнаємося, скільки було часу 10 годин 15 хвилин тому, тобто фактично нам треба відняти з поточного часу 10 годин і 15 хвилин:

```
від datetime import timedelta, datetime

now = datetime.now()
till_ten_hours_fifteen_minutes = now -
timedelta(hours=10, minutes=15)

print(till_ten_hours_fifteen_minutes)
```

Властивості timedelta

Клас timedelta має кілька властивостей, за допомогою яких ми можемо отримати часовий проміжок:

- `days`: повертає кількість днів
- `seconds`: повертає кількість секунд
- `microseconds`: повертає кількість мікросекунд.

Крім того, метод total_seconds() повертає загальну кількість секунд, куди входять і дні, і секунди, і мікросекунди.

Наприклад, дізнаємося якийсь тимчасовий період між двома датами:

```
fromdatetime import timedelta, datetime

now = datetime.now()
twenty_two_may = datetime(2017, 5, 22)
period = twenty_two_may - now
print("{} днів {} секунд {} мікросекунд".
format(period.days, period.seconds,
period.microseconds))
```

```
# 18 днів 17537 секунд 72765 мікросекунд

print("Всього: {} секунд".
format(period.total_seconds()))

# Всього: 1572737.072765 секунд
```

Порівняння дат

Так само як і рядки та числа, дати можна порівнювати за допомогою стандартних операторів порівняння:

```
від datetime import datetime

now = datetime.now()
deadline = datetime(2017, 5, 22)
if now > deadline:
    print("Термін здачі проекту пройшов")
elif now.day == deadline.day and now.month ==
deadline.month and now.year == deadline.year:
    print("Термін здачі проекту сьогодні")
else:
    period = deadline - now
    print("Залишилось {} днів".format(period.days))
```

XX. Модуль logging

Уявіть ситуацію, коли необхідно зберегти деякі налагоджувальні або інші важливі повідомлення де-небудь, щоб мати можливість пізніше перевірити, чи програма відпрацювала, як очікувалося. Як “зберегти десь” ці повідомлення?

Зробити це можна за допомогою модуля logging. (за матеріалами https://wombat.org.ua/AByteOfPython/standard_library.html)

```
importos,platform, logging

ifplatform.platform().startswith('Windows'):
   ogging_file = os.path.join(os.getenv('HOMEDRIVE'), \
                              os.getenv('HOMEPATH'), \
                              'test.log')
else:
    logging_file = os.path.join(os.getenv('HOME'),
'test.log')

print("Зберігаємо лог в",logging_file)

logging.basicConfig(
    level=logging.DEBUG,
```

```
    format='%(asctime)s : %(levelname)s : %(message)s',
    filename = logging_file,
    filemode = 'w',)

logging.debug("Початок програми")
logging.info("Якісь дії")
logging.warning("Програма вмирає")
```

Висновок:

$python3 use_logging.py

Зберігаємо лог в C:\Users\bsu\test.log

Якщо відкрити файл test.log, він виглядатиме приблизно так:

```
2018-03-06 12:31:35,033 : DEBUG : Початок програми
2018-03-06 12:31:35,033 : INFO : Якісь дії
2018-03-06 12:31:35,033 : WARNING : Програма вмирає
```

Як це працює:

Використовувалися три модулі зі стандартної бібліотеки: модуль os для взаємодії з операційною системою, модуль platform для отримання інформації про платформу (тобто операційну систему) та модуль logging для збереження лога.

Насамперед, за допомогою рядка, що повертається функцією platform.platform(), ми перевіряємо, яка операційна система використовується (для більш детальної інформації див. import platform; help(platform)). Якщо це Windows, то ми визначаємо диск, що містить домашній каталог, шлях до домашнього каталогу на ньому та ім'я файлу, в якому хочемо зберегти інформацію. Склавши всі ці три частини, ми отримуємо повний шлях до файлу. Для інших платформ нам потрібно знати шлях до домашнього каталогу користувача, і ми отримаємо повний шлях до файлу.

За допомогою функції os.path.join() ми поєднуємо три частини шляху до файлу разом. Ми використовуємо цю функцію замість простого об'єднання рядків, щоб гарантувати, що повний шлях до файлу записаний у форматі, що очікується операційною системою.

Далі ми конфігуруємо модуль logging таким чином, щоб він записував усі повідомлення у певному форматі у вказаний файл.

Нарешті, ми можемо виводити повідомлення, призначені для налагодження, інформування, попередження та навіть критичні повідомлення. Після виконання програми можна переглянути цей файл і дізнатися, що відбувалося в програмі, хоча користувачу, який запустив програму, нічого не було показано.

XXI. Створення GUI на Python за допомогою бібліотеки Tkinter

(за матеріалами сайту https://younglinux.info/tkinter/text.php)

Введення в tkinter

У різноманітті програм, які пишуть програмісти, виділяють додатки з графічним інтерфейсом користувача (GUI). При створенні таких програм стають важливими не лише алгоритми обробки даних, але й розробка для користувача програми зручного інтерфейсу, взаємодіючи з яким він визначатиме поведінку програми.

Сучасний користувач в основному взаємодіє з програмою за допомогою різних кнопок, меню, значків, вводячи інформацію в спеціальні поля, вибираючи певні значення у списках і т. д. Ці "зображення" у певному сенсі і формують GUI, надалі ми їх називатимемо віджетами (від англ. widget - "шту".

Для мови програмування Python такі віджети включені до спеціальної бібліотеки - tkinter. Якщо її імпортувати в програму (скрипт), можна користуватися її компонентами, створюючи графічний інтерфейс.

Послідовність кроків під час створення графічного застосування має свої особливості. Програма має виконувати своє основне призначення, бути зручною для користувача, реагувати на його дії. Ми не будемо вдаватися до подробиць розробки, а розглянемо які етапи приблизно потрібно пройти при програмуванні, щоб отримати програму з GUI:

1. Імпорт бібліотеки
2. Створення головного вікна
3. Створення віджет
4. Встановлення їх властивостей
5. Визначення подій
6. Визначення обробників подій
7. Розташування віджет на головному вікні
8. Відображення головного вікна

Імпорт модуля tkinter

Як і будь-який модуль, tkinter у Python можна імпортувати двома способами: командами import tkinter або from tkinter import *.

Надалі ми користуватимемося лише другим способом, тому що це дозволить не вказувати щоразу ім'я модуля при зверненні до об'єктів, які в ньому містяться. Слід звернути увагу, що у версії Python 3 ім'я модуля пишеться з малої літери (tkinter), хоча в попередніх версіях використовувалася велика (Tkinter). Отже, перший рядок програми має виглядати так:

```
з tkinter import *
```

Створення головного вікна

У сучасних операційних системах будь-який додаток користувача укладено у вікно, яке можна назвати головним, т.к. у ньому розташовуються решта віджетів. Об'єкт верхнього рівня вікна створюється при зверненні до класу Tk модуля tkinter. Змінну пов'язану з об'єктом-вікном прийнято

називати root (хоча зрозуміло, що можна назвати як завгодно, але вже прийнято). Другий рядок коду:

```
root = Tk()
```

Створення віджет

Допустимо у вікні розташовуватиметься лише одна кнопка. Кнопка створюється при зверненні до класу Button tkinter модуля. Об'єкт кнопка зв'язується з якоюсь змінною. У класу Button (як і решти класів, крім Tk) є обов'язковий параметр — об'єкт, якому кнопка належить (кнопка неспроможна " бути нічийної "). Поки ми маємо єдине вікно (root), воно і буде аргументом, що передається в клас при створенні об'єкта-кнопки:

```
but = Button(root)
```

Встановлення властивостей віджет

Кнопка має багато властивостей: розмір, колір фону та написи та ін. Ми розглянемо їх на наступному уроці. Поки ж встановимо лише одну властивість — текст напису (text):

```
but["text"] = "Друк"
```

Визначення подій та їх обробників

Різноманітність подій та способів їхньої обробки буде розглянуто на наступних уроках. Тут же просто торкнемося цього питання у зв'язку з потребою.

Що ж робитиме кнопка і в який момент вона це робитиме? Припустимо, що завдання кнопки вивести якесь повідомлення у потік виведення, використовуючи функцію print. Робити вона це при натисканні на неї лівою кнопкою миші.

Дії (алгоритм), які відбуваються при тій чи іншій події, можуть бути досить складними. Тому часто їх оформляють як функції, а потім викликають, коли вони знадобляться. Нехай у нас друк на екран буде оформлений у вигляді функції printer:

```
def printer(event):
    print ("Як завжди черговий 'Hello World!'")
```

Не забувайте, що функцію бажано (майже обов'язково) розміщувати на початку коду. Параметр event – це подія.

Подія натискання лівою кнопкою миші виглядає так: <Button-1>. Потрібно пов'язати цю подію з обробником (функцією printer). Для зв'язку призначено метод bind. Синтаксис зв'язування події з обробником виглядає так:

```
but.bind("<Button-1>",printer)
```

Подія - мав розмір вікна:

```
# обробник res закриття вікна
root.bind('<Configure>',res)
```

В обробнику можна отримати новий розмір графічного вікна:

```
  str=root.geometry()
  k1=str.index('x')
```

```
    xmax1=str[:k1]
    k2=str.index('+',k1+1)
    ymax1=str[k1+1:k2]
    xmax = int (xmax1)
    ymax=int(ymax1)
```

Подія – закрити вікно:

```
# обробник window_deleted закриття вікна
root.protocol('WM_DELETE_WINDOW', window_deleted)
```

Розміщення віджет

Якщо ви помітите, то в будь-якому додатку віджети не розкидані по вікну абияк, а добре організовані, інтерфейс продуманий до дрібниць і зазвичай підпорядкований певним стандартам. До стандартів нам далеко, потрібно просто кнопку якось відобразити у вікні. Найпростіший спосіб – це використання методу pack.

```
but.pack()
```

Якщо не вставити цей рядок коду, то кнопка у вікні так і не з'явиться, хоча вона є в програмі.

Відображення головного вікна

Ну і нарешті, головне вікно теж не з'явиться, доки не буде викликаний спеціальний метод mainloop:

```
root.mainloop()
```

Цей рядок коду повинен бути завжди в кінці скрипту!

У результаті код програми може виглядати таким чином:

```
fromtkinter import *

defprinter(event):
     print ("Як завжди черговий 'Hello World!'")

root = Tk()
but = Button(root)
but["text"] = "Друк"
but.bind("<Button-1>",printer)

but.pack()
root.mainloop()
```

При програмуванні графічного інтерфейсу користувача ефективнішим виявляється об'єктно-орієнтований підхід. Тому багато «речей»

оформляються у вигляді класів. У нашому прикладі можна також використовувати клас:

```
fromtkinter import *

classBut_print:
     def __init__(self):
          self.but = Button(root)
          self.but["text"] = "Друк"
          self.but.bind("<Button-1>",self.printer)
          self.but.pack()
     def printer(self,event):
          print ("Як завжди черговий 'Hello World!'")

root = Tk()
obj = But_print()
root.mainloop()
```

Розмітка віджетів у Tkinter — pack, grid та place

Познайомимось із менеджерами розмітки. Коли ми створюємо графічний інтерфейс нашої програми, ми визначаємо, які віджети будемо використовувати і як вони будуть розташовані в додатку. Щоб організувати віджети у додатку, використовуються спеціальні невидимі об'єкти – менеджери розмітки.

Існує два види віджетів:

- контейнери;
- дочірні віджети.

Контейнери поєднують віджети для формування розмітки. У Tkinter є три вбудовані менеджери розмітки: pack, grid і place.

- **Place**– це менеджер геометрії, який розміщує віджети, використовуючи абсолютну позиціонування.
- **Pack**– це менеджер геометрії, який розміщує віджети по горизонталі та вертикалі.
- **Grid**– це менеджер геометрії, який розміщує віджети у двовимірній сітці.

Метод place() у Tkinter — Абсолютне позиціонування

Найчастіше розробникам потрібно використовувати менеджери розмітки. Є кілька ситуацій, у яких слід використати саме абсолютне позиціонування. В рамках абсолютного позиціонування розробник визначає позицію та розмір кожного віджету в пікселях. Під час зміни розмірів вікна розмір та позиція віджетів не змінюються.

Зображення для прикладу:bardejov.jpg rotunda.jpg mincol.jpgЗбережіть у папці поруч із файлом absolute.py код для якого буде нижче.

Для цього скрипта необхідно встановити пакет Image:

```
python -m pip install Image
```

Таким чином, на різних платформах програми виглядають по-різному. Те, що виглядає нормально на Linux, може некоректно відображатися на Mac OS. Зміна шрифтів у нашому додатку може також зіпсувати розмітку. Якщо ми переведемо нашу програму іншою мовою, ми повинні доопрацювати і розмітку.

```
absolute.py
від PIL import Image, ImageTk
з tinter import Tk, BOTH
від tkinter.ttk import Frame, Label, Style

class Example(Frame):

def __init__(self):
super().__init__()

self.initUI()


def initUI(self):

self.master.title("Absolute positioning")
self.pack(fill=BOTH, expand=1)

Style().configure("TFrame", background="#333")

bard = Image.open("bardejov.jpg")
bardejov = ImageTk.PhotoImage(bard)
label1 = Label(self, image=bardejov)
label1.image = bardejov
label1.place(x=20, y=20)

rot = Image.open("rotunda.jpg")
rotunda = ImageTk.PhotoImage(rot)
label2 = Label(self, image=rotunda)
label2.image = rotunda
label2.place(x=40, y=160)

minc = Image.open("mincol.jpg")
mincol = ImageTk.PhotoImage(minc)
label3 = Label(self, image=mincol)
```

```
label3.image = mincol
label3.place(x=170, y=50)


def main():

root = Tk()
root.geometry("300x280+300+300")
app = Example()
root.mainloop()


if __name__ == '__main__':
main()
```

У цьому прикладі ми розташували три зображення за допомогою абсолютного позиціонування. Ми використали менеджер геометрії place.

```
від PIL import Image, ImageTk
```

Ми використовували Image та ImageTk з модуля PIL (Python Imaging Library).

```
style = Style()
style.configure("TFrame", background="#333")
```

За допомогою стилів ми змінили фон нашого вікна на темно-сірий.

```
bard = Image.open("bardejov.jpg")
bardejov = ImageTk.PhotoImage(bard)
```

Ми створили об'єкт зображення та об'єкт фото зображення зі збережених раніше зображень у поточній робочій директорії.

```
label1 = Label(self, image=bardejov)
```

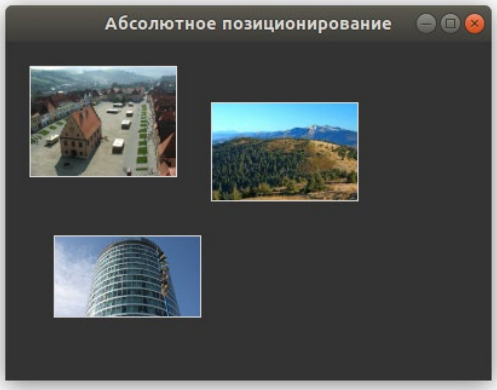
Ми створили Label (ярлик) із зображенням. Дані ярлики можуть містити зображення і текст.

```
label1.image = bardejov
```

Нам потрібно зберегти посилання на зображення, щоб не втратити його, якщо збирач сміття (Garbage collector) його не закриє.

```
label1.place(x=20, y=20)
```

Ярлик розміщений у рамці за координатами x=20 та y=20.



Tkinter pack() — розміщення віджетів по горизонталі та вертикалі.

Менеджер геометрії pack() упорядковує віджети у горизонтальні та вертикальні блоки. Макетом можна керувати за допомогою параметрів fill, expand та side.

Приклад створення кнопок у Tkinter

У наступному прикладі ми розмістимо дві кнопки в правому нижньому кутку нашого вікна. Для цього ми скористаємося менеджером pack.

```
з tinter import Tk, RIGHT, BOTH, RAISED
від tkinter.ttk import Frame, Button, Style


class Example(Frame):

def __init__(self):
super().__init__()
self.initUI()

def initUI(self):
self.master.title("Кнопки в kinter")
self.style = Style()
self.style.theme_use("default")

frame = Frame(self, relief=RAISED, borderwidth=1)
frame.pack(fill=BOTH, expand=True)

self.pack(fill=BOTH, expand=True)

closeButton = Button(self, text="Закрити")
closeButton.pack(side=RIGHT, padx=5, pady=5)
okButton = Button(self, text="Готово")
okButton.pack(side=RIGHT)
```

```
def main():

root = Tk()
root.geometry("300x200+300+300")
app = Example()
root.mainloop()


if __name__ == '__main__':
main()
```

Ми маємо дві рамки. Перша рамка – основна, а також друга – додаткова, яка розтягується в обидва боки та зсуває дві кнопки у нижню частину основної рамки. Кнопки знаходяться у горизонтальному контейнері та розміщені у її правій частині.

```
frame = Frame(self, relief=RAISED, borderwidth=1)
frame.pack(fill=BOTH, expand=True)
```

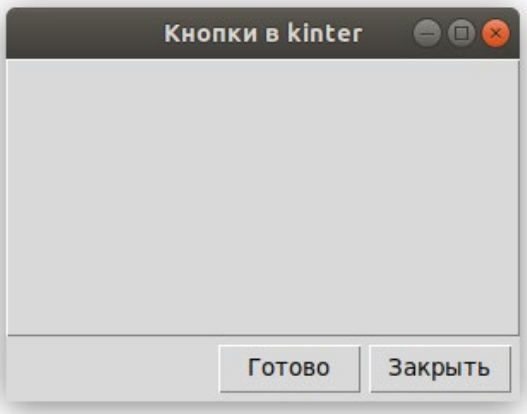
Ми створили ще один віджет Frame. Цей віджет займає практично весь простір вікна. Ми змінюємо межі рамки, щоб сама рамка була видна. За умовчанням вона пласка.

```
closeButton = Button(self, text="Закрити")
closeButton.pack(side=RIGHT, padx=5, pady=5)
```

Кнопка closeButton створена. Вона розташована у горизонтальному контейнері. Параметр side дозволяє помістити кнопку у правій частині горизонтальної смуги. Параметри padx та pady дозволяють встановити відступ між віджетами. Параметр padx встановлює простір між віджетами кнопки closeButton та правою межею кореневого вікна.

```
okButton.pack(side=RIGHT)
```

Кнопка okButton розміщена біля closeButton із встановленим відступом (padding) у 5 пікселів.



Створюємо програму для відгуків на Tkinter

Менеджер pack- Це простий менеджер розмітки. Його можна використовувати для найпростіших завдань розмітки. Щоб створити складнішу розмітку, необхідно використовувати більше рамок, кожна з яких має власний менеджер розмітки.

review.py

```
з tinter import Tk, Text, BOTH, X, N, LEFT
з tkinter.ttk import Frame, Label, Entry


class Example(Frame):

def __init__(self):
super().__init__()
self.initUI()

def initUI(self):
self.master.title("Залишити відгук")
self.pack(fill=BOTH, expand=True)

frame1 = Frame(self)
frame1.pack(fill=X)

lbl1 = Label(frame1, text="Заголовок", width=10)
lbl1.pack(side=LEFT, padx=5, pady=5)

entry1 = Entry(frame1)
entry1.pack(fill=X, padx=5, expand=True)

frame2 = Frame(self)
frame2.pack(fill=X)

lbl2 = Label(frame2, text="Автор", width=10)
lbl2.pack(side=LEFT, padx=5, pady=5)

entry2 = Entry(frame2)
entry2.pack(fill=X, padx=5, expand=True)

frame3 = Frame(self)
frame3.pack(fill=BOTH, expand=True)

lbl3 = Label(frame3, text="Відгук", width=10)
lbl3.pack(side=LEFT, anchor=N, padx=5, pady=5)

txt = Text(frame3)
txt.pack(fill=BOTH, pady=5, padx=5, expand=True)


def main():

root = Tk()
root.geometry("300x300+300+300")
app = Example()
root.mainloop()
```

```
if __name__ == '__main__':
main()
```

На цьому прикладі видно, як можна створити складнішу розмітку з численними рамками та менеджерами pack().

```
1
self.pack(fill=BOTH, expand=True)
```

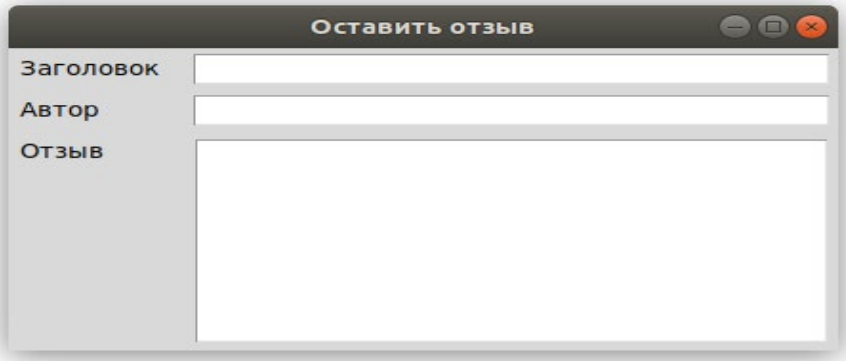
Перша рамка є базовою. На ній розташовуються всі інші рамки. Варто зазначити, що навіть при організації дочірніх віджетів у рамках ми керуємо ними на базовій рамці.

```
frame1 = Frame(self)
frame1.pack(fill=X)

lbl1 = Label(frame1, text="Заголовок", width=10)
lbl1.pack(side=LEFT, padx=5, pady=5)

entry1 = Entry(frame1)
entry1.pack(fill=X, padx=5, expand=True)
```

Перші два віджети розміщені на першій рамці. Поле для введення даних розтягнуте горизонтально з параметрами fill та expand.

```
frame3 = Frame(self)
frame3.pack(fill=BOTH, expand=True)

lbl3 = Label(frame3, text="Відгук", width=10)
lbl3.pack(side=LEFT, anchor=N, padx=5, pady=5)

txt = Text(frame3)
txt.pack(fill=BOTH, pady=5, padx=5, expand=True)
```

У третій рамці ми розмістили ярлик та віджет для введення тексту. Ярлик закріплений по північній стороні anchor=N, а віджет тексту займає решту простору.



Розмітка grid() у Tkinter для створення калькулятора

Менеджер геометрії grid() Tkinter використовується для створення сітки кнопок для калькулятора.

calculator.py

```
з tinter import Tk, W, E
від tkinter.ttk import Frame, Button, Entry, Style


class Example(Frame):

def __init__(self):
super().__init__()
self.initUI()

def initUI(self):
self.master.title("Калькулятор на Tkinter")

Style().configure("TButton", padding=(0, 5, 0, 5),
font='serif 10')

self.columnconfigure(0, pad=3)
self.columnconfigure(1, pad=3)
self.columnconfigure(2, pad=3)
self.columnconfigure(3, pad=3)

self.rowconfigure(0, pad=3)
self.rowconfigure(1, pad=3)
self.rowconfigure(2, pad=3)
self.rowconfigure(3, pad=3)
self.rowconfigure(4, pad=3)

entry = Entry(self)
entry.grid(row=0, columnspan=4, sticky=W+E)
cls = Button(self, text="Очистити")
cls.grid(row=1, column=0)
bck = Button(self, text="Видалити")
bck.grid(row=1, column=1)
lbl = Button(self)
lbl.grid(row=1, column=2)
clo = Button(self, text="Закрити")
clo.grid(row=1, column=3)
sev = Button(self, text="7")
sev.grid(row=2, column=0)
eig = Button(self, text="8")
eig.grid(row=2, column=1)
nin = Button (self, text = "9")
nin.grid(row=2, column=2)
div = Button(self, text="/")
div.grid(row=2, column=3)

fou = Button(self, text="4")
```

```
fou.grid(row=3, column=0)
fiv = Button (self, text = "5")
fiv.grid(row=3, column=1)
six = Button (self, text = "6")
six.grid(row=3, column=2)
mul = Button(self, text="*")
mul.grid(row=3, column=3)

one = Button(self, text="1")
one.grid(row=4, column=0)
two = Button(self, text="2")
two.grid(row=4, column=1)
thr = Button (self, text = "3")
thr.grid(row=4, column=2)
mns = Button(self, text="-")
mns.grid(row=4, column=3)

zer = Button(self, text="0")
zer.grid(row=5, column=0)
dot = Button(self, text=".")
dot.grid(row=5, column=1)
equ = Button(self, text="=")
equ.grid(row=5, column=2)
pls = Button(self, text="+")
pls.grid(row=5, column=3)

self.pack()


def main():
root = Tk()
app = Example()
root.mainloop()


if __name__ == '__main__':
main()
```

Менеджер grid() використовується для організації кнопок у контейнері рамки.

```
Style().configure("TButton", padding=(0, 5, 0, 5),
font='serif 10')
```

Ми налаштували віджет кнопки так, щоб відображався специфічний шрифт та застосовувався відступ (padding) у 3 пікселі.

```
self.columnconfigure(0, pad=3)
...
self.rowconfigure(0, pad=3)
```

Ми використовували методи columnconfigure() та rowconfigure(), щоб створити певний простір у сітці рядків та стовпців. Завдяки цьому кроку ми розділяємо кнопки певним порожнім простором.

```
entry = Entry(self)
entry.grid(row=0, columnspan=4, sticky=W+E)
```

Віджет графи введення– це місце, де відображатимуться цифри. Цей віджет розташований у першому ряду та охоплює всі чотири стовпці. Віджети можуть не займати весь простір, який виділяється клітинами у створеній сітці.

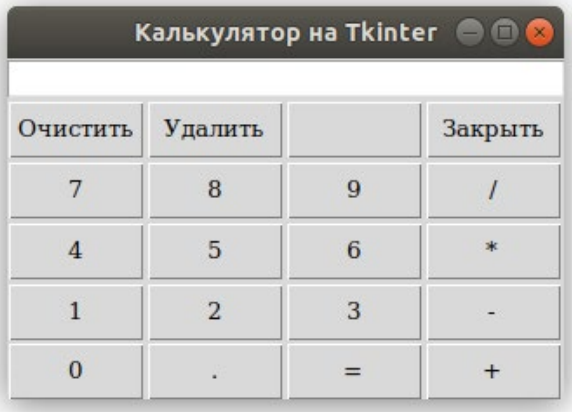
Параметр sticky розширює віджет у вказаному напрямку. У нашому випадку ми можемо переконатися, що наш віджет графи введення був розширений зліва направо W+E (схід-захід).

```
cls = Button(self, text="Очистити")
cls.grid(row=1, column=0)
```

Кнопка очищеннявстановлено у другому рядку та першому стовпці. Варто зазначити, що рядки та стовпці починаються з нуля.

```
self.pack()
```

Метод pack() показує віджет рамки та дає їй початковий розмір. Якщо додаткові параметри не вказуються, розмір буде таким, щоб усі дочірні віджети могли поміститися. Цей метод компонує віджет рамки у верхньому кореневому вікні, що також є контейнером. Менеджер grid() використовується для організації кнопок у віджеті рамки.



Приклад створення діалогового вікна в Tkinter

Наступний приклад створює діалогове вікно за допомогою менеджера геометрії grid.

```
windows.py

з tinter import Tk, Text, BOTH, W, N, E, S
від tkinter.ttk import Frame, Button, Label, Style


class Example(Frame):

def __init__(self):
```

```
super().__init__()
self.initUI()


def initUI(self):

self.master.title("Діалогове вікно в Tkinter")
self.pack(fill=BOTH, expand=True)

self.columnconfigure(1, weight=1)
self.columnconfigure(3, pad=7)
self.rowconfigure(3, weight=1)
self.rowconfigure(5, pad=7)

lbl = Label(self, text="Вікна")
lbl.grid (sticky = W, pady = 4, padx = 5)

area = Text(self)
area.grid(row=1, column=0, columnspan=2, rowspan=4, padx=5,
sticky=E+W+S+N)

abtn = Button(self, text="Актив.")
abtn.grid(row=1, column=3)

cbtn = Button(self, text="Закрити")
cbtn.grid(row=2, column=3, pady=4)

hbtn = Button(self, text="Допомога")
hbtn.grid(row=5, column=0, padx=5)

obtn = Button (self, text = "Готово")
obtn.grid(row=5, column=3)


def main():

root = Tk()
root.geometry("350x300+300+300")
app = Example()
root.mainloop()


if __name__ == '__main__':
main()
```

У цьому прикладі ми використовували віджет ярлика, текстовий віджет та чотири кнопки.

```
self.columnconfigure(1, weight=1)
self.columnconfigure(3, pad=7)
self.rowconfigure(3, weight=1)
self.rowconfigure(5, pad=7)
```

Ми додали невеликий простір між віджетами у сітці. Параметр weight створює можливість розширення другого стовпця та четвертого ряду. У цьому рядку і стовпці знаходиться текстовий віджет, тому простір, що залишився, заповнює цей віджет.

```
lbl = Label(self, text="Вікна")
lbl.grid (sticky = W, pady = 4, padx = 5)
```

Віджет ярликатакож створюється та поміщається у сітку. Якщо не вказуються ряд і стовпець, він займе перший ряд і стовпець. Ярлик закріплюється біля західної частини вікна sticky=W і має певні відступи навколо кордонів.

```
area = Text(self)
area.grid(row=1, column=0, columnspan=2, rowspan=4,
padx=5, sticky=E+W+S+N)
```

Створюється текстовий віджет і поміщається у другий рядок і перший стовпець. Він охоплює два стовпці та чотири рядки.

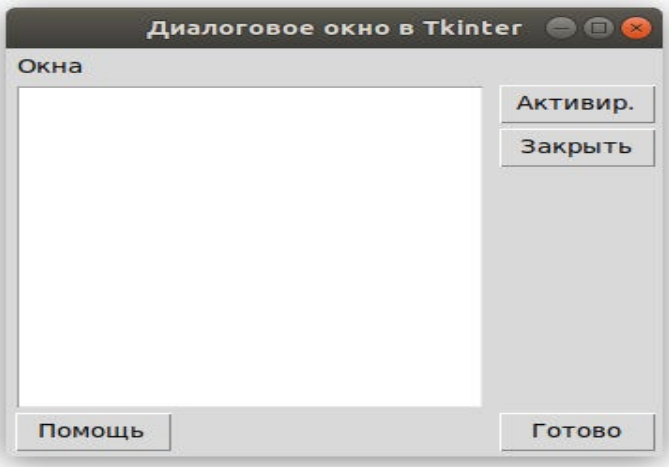
Між віджетом і лівим краєм кореневого вікна є простір у 4 пікселі. Також віджет закріплений біля всіх чотирьох сторін. Тому, коли вікно розширюється, віджети текстів збільшуються у всіх напрямках.

```
abtn = Button(self, text="Актив.")
abtn.grid(row=1, column=3)

cbtn = Button(self, text="Закрити")
cbtn.grid(row=2, column=3, pady=4)
```

Ці дві кнопки знаходяться біля текстового віджету.

```
hbtn = Button(self, text="Допомога")
hbtn.grid(row=5, column=0, padx=5)

obtn = Button (self, text = "Готово")
obtn.grid(row=5, column=3)
```

Ці дві кнопки знаходяться під текстовим віджетом. Кнопка «Допомога» розташована в першому стовпчику, а кнопка «Готово» в останньому стовпчику.



XXII. Віджети (графічні об'єкти) та їх властивості.

Розглянемо частину графічних об'єктів (віджет), які у бібліотеці Tkinter: кнопки, поля для введення, мітки, прапорці, перемикачі і списки. Слід розуміти, що графічний інтерфейс користувача досить стандартний, і тому будь-які подібні бібліотеки-модулі (в тому числі і Tkinter) містять приблизно однакові віджети.

Кожен клас віджет має певні властивості, значення яких можна задавати під час їх створення, а також програмувати їх зміну при дії користувача та в результаті виконання програми.

Кнопки

Об'єкт-кнопка створюється викликом класу Button tkinter. При цьому обов'язковим аргументом є лише батьківський віджет (наприклад вікно верхнього рівня). Інші властивості можуть вказуватися під час створення кнопки або задаватися (змінюватися) пізніше. Синтаксис:

змінна = Button (родить_віджет, [властивість = значення, … ….])

Кнопка має багато властивостей, у прикладі нижче вказані лише деякі з них.

```
з tkinter import *

root = Tk()

but = Button(root,
           text="Це кнопка", #напис на кнопці
           width=30,height=5, #ширина та висота
           bg="white",fg="blue") #колір фону та написи

but.pack()
```

```
root.mainloop()
```

bg та fg – це скорочення від background (фон) та foreground (передній план). Ширина та висота вимірюються в знайоместах (кількість символів).

Мітки

Мітки (або написи) — це досить прості віджети, що містять рядок (або кілька рядків) тексту та службовці, в основному, для інформування користувача.

```
lab = Label(root, text="Це мітка! \n З двох рядків.",
font="Arial 18")
```

Однорядкове текстове поле

Таке поле створюється викликом класу модуля Entry tkinter. Користувач може ввести лише один рядок тексту.

```
ent = Entry (root, width = 20, bd = 3)
```

bd- Це скорочення від межіширини (ширина кордону).

Елемент Entry є полем для введення тексту. Конструктор Entry приймає такі параметри:

```
Entry(master, options)
```

Де master – посилання на батьківське вікно, а options – набір наступних параметрів:

- bg: фоновий колір
- bd: товщина кордону
- cursor: курсор покажчика миші при наведенні на текстове поле
- fg: колір тексту
- font: шрифт тексту
- justify: встановлює вирівнювання тексту Значення LEFT вирівнює текст по лівому краю, CENTER - по центру, RIGHT - по правому краю
- relief: визначає тип кордону, за замовчуванням значення FLAT
- selectbackground: фоновий колір виділеного шматка тексту.
- selectforeground: колір виділеного тексту
- show: задає маску для символів, що вводяться.
- state: стан елемента може приймати значення NORMAL (за замовчуванням) і DISABLED
- textvariable: встановлює прив'язку до елемента StringVar
- width: ширина елемента

Визначимо елемент Entry та за натисканням на кнопку виведемо його текст в окреме вікно з повідомленням:

```
fromtkinter import *
fromtkinter import messagebox
```

```
defshow_message():
messagebox.showinfo("GUI Python", message.get())

root =Tk()
root.title("GUI на Python")
root.geometry("300x250")

message =StringVar()

message_entry =Entry(textvariable=message)
message_entry.place(relx=.5, rely=.1, anchor="c")

message_button =Button(text="Click Me", command=show_message)
message_button.place(relx=.5, rely=.5, anchor="c")

root.mainloop()
```

Для виведення повідомлення тут застосовується додатковий модуль messagebox, що містить функцію showinfo(), яка і виводить введений у текстове поле текст. Для отримання введеного тексту використовується комп'ютер StringVar.

Тепер створимо складніший приклад із формою введення:

```
fromtkinter import *
fromtkinter import messagebox


defdisplay_full_name():
messagebox.showinfo("GUI Python", name.get() +" " + surname.get())

root =Tk()
root.title("GUI на Python")

name =StringVar()
surname =StringVar()

name_label =Label(text="Введіть ім'я:")
surname_label =Label(text="Введіть прізвище:")

name_label.grid(row=0, column=0, sticky="w")
surname_label.grid(row=1, column=0, sticky="w")

name_entry =Entry(textvariable=name)
surname_entry =Entry(textvariable=surname)

name_entry.grid(row=0,column=1, padx=5, pady=5)
surname_entry.grid(row=1,column=1, padx=5, pady=5)


message_button =Button(text="Click Me", command=display_full_name)
message_button.grid(row=2,column=1, padx=5, pady=5, sticky="e")

root.mainloop()
```

Методи Entry

Елемент Entry має низку методів. Основні з них:

- insert(index, str): вставляє в текстове поле рядок за певним індексом
- get(): повертає введений текстове поле текст
- delete(first, last=None): видаляє символ за індексом first. Якщо вказано параметр last, видалення проводиться до індексу last. Щоб видалити до кінця, в якості другого параметра можна використовувати значення END.

Використовуємо методи у програмі:

```
fromtkinter import *
fromtkinter import messagebox


defclear():
name_entry.delete(0, END)
surname_entry.delete(0, END)


defdisplay():
messagebox.showinfo("GUI Python", name_entry.get() +" " +
surname_entry.get())

root =Tk()
root.title("GUI на Python")

name_label =Label(text="Введіть ім'я:")
surname_label =Label(text="Введіть прізвище:")

name_label.grid(row=0, column=0, sticky="w")
surname_label.grid(row=1, column=0, sticky="w")

name_entry =Entry()
surname_entry =Entry()

name_entry.grid(row=0,column=1, padx=5, pady=5)
surname_entry.grid(row=1,column=1, padx=5, pady=5)

# Вставка початкових даних
name_entry.insert(0, "Tom")
surname_entry.insert(0, "Soyer")

display_button =Button(text="Display", command=display)
clear_button =Button(text="Clear", command=clear)

display_button.grid(row=2, column=0, padx=5, pady=5, sticky="e")
clear_button.grid(row=2, column=1, padx=5, pady=5, sticky="e")
```

```
root.mainloop()
```

При запуску програми відразу в обидва текстові поля додається текст за замовчуванням:

```
name_entry.insert(0, "Tom")
surname_entry.insert(0, "Soyer")

name_entry.insert("end", "Tom")
```

А так можна вставити текст «наприкінці поля»

Кнопка Clear очищає обидва поля, викликаючи метод delete:

```
def clear():
name_entry.delete(0, END)
surname_entry.delete(0, END)
```

Друга кнопка, використовуючи метод get, отримує введений текст:

```
def display():
messagebox.showinfo("GUI Python", name_entry.get() + "
" + surname_entry.get())
```

Причому, як видно з прикладу, нам необов'язково звертатися до тексту в Entry через змінні типу StringVar, ми можемо зробити це безпосередньо через метод get

Багаторядкове текстове поле

Text призначений для надання користувачеві можливості введення не одного рядка тексту, а значно більше.

```
tex = Text (root, width = 40,
          font="Verdana 12",
          wrap = WORD)
```

Остання властивість (wrap) в залежності від свого значення дозволяє переносити текст, що вводиться користувачем або за символами, або за словами, або взагалі не переносити, доки користувач не натисне Enter.

Радіокнопки (перемикачі)

Об'єкт-радіокнопка ніколи не використовується по одному. Їх використовують групами, при цьому в одній групі може бути включена лише одна кнопка.

```
var=IntVar()
var.set(1)
rad0 = Radiobutton(root,text="Перша",
          variable=var,value=0)
rad1 = Radiobutton(root,text="Друга",
          variable=var,value=1)
rad2 = Radiobutton(root,text="Третя",
```

```
            variable=var,value=2)
```

Одна група визначає значення однієї змінної, тобто якщо в прикладі буде обрана радіокнопка rad2, то змінної буде var буде 2. Спочатку також потрібно встановити значення змінної (вираз var.set(1) задає значення змінної var рівне 1).

Прапорці

Об'єкт checkbutton призначений для вибору не взаємовиключних пунктів у вікні (у групі можна активувати один, два або більше прапорців або один). На відміну від радіокнопок, значення кожного прапорця прив'язується до своєї змінної, значення якої визначається опціями onvalue (включено) та offvalue (вимкнено) в описі прапорця.

```
c1 = IntVar()
c2 = IntVar()
che1 = Checkbutton(root,text="Перший прапорець",
          variable=c1,onvalue=1,offvalue=0)
che2 = Checkbutton(root,text="Другий прапорець",
          variable=c2,onvalue=2,offvalue=0)
```

Списки

Виклик класу Listbox створює об'єкт, в якому користувач може вибрати один або кілька пунктів залежно від значення опції selectmode. У прикладі нижче значення SINGLE дозволяє вибирати лише один пункт зі списку.

```
r = ['Linux','Python','Tk','Tkinter']
lis = Listbox(root,selectmode=SINGLE,height=4)
for i in r:
     lis.insert(END,i)
```

Спочатку список (Listbox) порожній. За допомогою циклу for до нього додаються пункти зі списку (тип даних) r. Додавання відбувається за допомогою спеціального методу класу Listbox – insert. Даний метод приймає два параметри: куди додати та що додати.

Більшість методів різних віджет ми розглянемо під час вивчення даного курсу.

Віджети (графічні об'єкти) та їх властивості

Продовжимо розглядати графічні об'єкти (віджети), які у бібліотеці Tkinter. Це будуть рамка (frame), шкала (scale), смуга прокручування (scrollbar), вікно верхнього рівня (toplevel).

Frame (рамка)

Як з'ясується пізніше, рамки (фрейми) є гарним інструментом для організації інших віджет у групи всередині вікна, а також оформлення.

```
з tkinter import *

root = Tk()
```

```
fra1 = Frame(root,width=500,height=100,bg="darkred")
fra2 =

    Frame(root,width=300,height=200,bg="green",bd=20)
fra3 = Frame(root,width=500,height=150,bg="darkblue")

fra1.pack()
fra2.pack()
fra3.pack()

root.mainloop()
```

Цей скрипт створює три кадри різного розміру. Властивість bd (скорочення від boderwidth) визначає відстані від краю рамки до віджетів, що у неї віджетів (якщо вони є).

На фреймах можна розміщувати віджети як на основному вікні (root). Тут текстове поле знаходиться на рамці fra2.

```
ent1 = Entry(fra2,width=20)
ent1.pack()
```

Scale (шкала)

Призначення шкали – це надання користувачеві вибору певного значення з певного діапазону. Зовні шкала є горизонтальною або вертикальною смугою з розміткою, по якій користувач може пересувати движок, здійснюючи тим самим вибір значення.

```
sca1 = Scale(fra3,orient=HORIZONTAL,length=300,
          from_=0,to=100,tickinterval=10,resolution=5)
sca2 = Scale(root,orient=VERTICAL,length=400,
          from_=1,to=2,tickinterval=0.1,resolution=0.1)
```

Властивості:

- `orient`визначає напрямок шкали;
- `length`- Довжина шкали в пікселях;
- `from_` і to - з якого значення шкала починається і яким закінчується (тобто діапазон значень);
- `tickinterval`– інтервал, через який відображаються позначки для шкали;
- `resolution`- Мінімальна довжина відрізка, на яку користувач може пересунути двигун.

Scrollbar (смуга прокручування)

Цей віджет дозволяє прокручувати вміст іншого віджету (наприклад, текстового поля або списку). Прокручування може бути як по горизонталі, так і по вертикалі.

```
з tkinter import *

root = Tk()
```

```
tx = Text(root,width=40,height=3,font='14')
scr = Scrollbar(root,command=tx.yview)
tx.configure(yscrollcommand=scr.set)

tx.grid(row=0,column=0)
scr.grid(row=0,column=1)
root.mainloop()
```

У прикладі спочатку створюється текстове поле (tx), потім смуга прокручування (scr), яка прив'язується за допомогою опції command до поля tx вертикальної осі (yview). Далі поле tx змінюється (конфігурується) за допомогою методу configure: встановлюється значення опції yscrollcommand.

Тут використовується незнайомий нам поки ще метод grid, що є іншим способом розташування віджет на вікні.

Toplevel (вікно верхнього рівня)

За допомогою класу Toplevel створюються дочірні вікна, на яких також можуть розташовуватися віджети. Слід зазначити, що при закритті головного вікна (або батьківського) вікно Toplevel також закривається. З іншого боку, закриття дочірнього вікна не призводить до закриття головного вікна.

```
win = Toplevel (root, relief = SUNKEN, bd = 10, bg =
"lightblue")
win.title("Дочірнє вікно")
win.minsize(width=400,height=200)
```

Метод title визначає заголовок вікна. Метод minsize конфігурує мінімальний розмір вікна (є метод maxsize, що визначає максимальний розмір вікна). Якщо значення аргументів minsize буде таким самим, як у maxsize, то користувач не зможе змінювати розміри вікна.

XXIII. Програмування подій.

Метод bind модуля Tkinter.

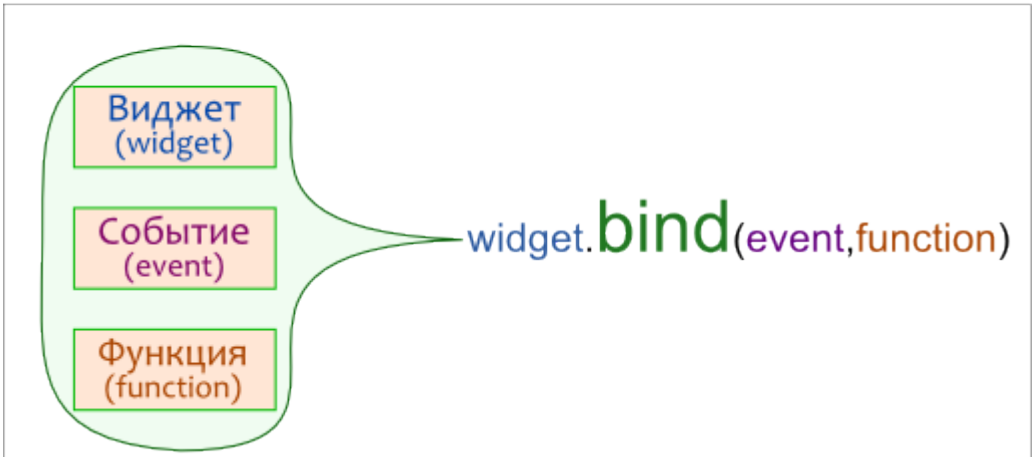
Програми з графічним інтерфейсом користувача (GUI) повинні не просто красиво відображатися на екрані, але й виконувати будь-які дії, реалізуючи потреби користувача. На минулих уроках було розказано як створити GUI, на цьому уроці розглянемо, як додати йому функціональність, тобто можливість здійснювати за його допомогою ті чи інші дії.

На відміну від консольних додатків, які зазвичай виконуються при мінімальних зовнішніх впливах, графічний додаток зазвичай чекає на будь-які зовнішні впливи (клацань кнопкою миші, натискання клавіш на клавіатурі, зміни віджетів) і потім виконує закладену програмістом дію. З такого принципу роботи можна вивести наступну схему налаштування функціональності GUI: на віджет щось «впливає» з ? виконується якась функція (дія). Зовнішній вплив на графічний компонент називається подією. Події досить багато (основний їх перелік ми розглянемо на наступному

занятті). На цьому занятті будемо використовувати лише два види подій: клацання лівою кнопкою миші () та натискання клавіші Enter ().

Одним із способів зв'язування віджету, події та функції (те, що має відбуватися після події) є використання методу bind. Синтаксис зв'язування представлений малюнку нижче.

Добавление функциональности графическому элементу с помощью метода bind



Розглянемо різні приклади додавання функціональності GUI.

приклад 1.

```
defoutput(event):
     s = ent.get()
     if s == "1":
          tex.delete(1.0,END)
          tex.insert(END,"Обслуговування клієнтів на
                                        другому
поверсі")
     elif s == "2":
          tex.delete(1.0,END)
          tex.insert(END,"Пластикові карти видають у
                                    сусідній будівлі")
     else:
          tex.delete(1.0,END)
          tex.insert(END,"Введіть 1 або 2 у поле
                                              зліва")

з tkinter import *
root = Tk()

ent = Entry(root,width=1) ## 16
but = Button(root,text="Вивести") ## 17
tex =Text(root,width=20,height=3,font="12",wrap=WORD)

ent.grid(row=0,column=0,padx=20) ## 22
but.grid(row=0,column=1)
tex.grid(row=0,column=2,padx=20,pady=10)
```

```
but.bind("<Button-1>",output) ## 24

root.mainloop()
```

Розглянемо код, починаючи з 16-го рядка.

У рядках 16-18 створюються три віджети: однорядкове текстове поле, кнопка та багаторядкове текстове поле. У першому полі користувач повинен щось ввести, потім натиснути кнопку і отримати відповідь у другому полі.

У рядках 20-22 використовується менеджер grid для розміщення віджетів. Властивості padx та pady визначають кількість пікселів від віджету до краю рамки (або комірки) по осях x та y відповідно.

У стоку 24 саме відбувається зв'язування кнопки з подією натискання лівої кнопки миші і функцією output. Всі ці три компоненти (віджет, подія та функція) зв'язуються за допомогою методу bind. В даному випадку при натисканні лівою кнопкою миші по кнопці but буде викликана функція output.

Отже, якщо раптом користувач клацне лівою кнопкою миші по кнопці, виконається функція output (ні в якому іншому випадку вона виконуватися не буде). Ця функція (рядки 1-11) виводить інформацію у друге текстове поле. Яку саме інформацію залежить від того, що користувач ввів у перше текстове поле. Як аргумент функції передається подія (у разі).

Усередині гілок if-elif-else використовуються методи delete та insert. Перший видаляє символи з текстового поля, другий — вставляє. 1.0 - позначає перший рядок, перший символ (нумерація символів починається з нуля).

приклад 2.

```
li = ["red","green"]
defcolor(event):
     fra.configure(bg=li[0])
     li[0],li[1] = li[1],li[0]

defoutgo(event):
     root.destroy()

з tkinter import *
root = Tk()
fra = Frame (root, width = 100, height = 100) # 12 & 13
but = Button(root,text="Вихід")
fra.pack()
but.pack()
root.bind("<Return>",color) # 18
but.bind("<Button-1>",outgo) # 19

root.mainloop()
```

Тут створюються два віджети (рядки 12, 13): кадр і кнопка.

Програма реагує на дві події: натискання клавіші Enter в межах головного вікна (рядок 18) та натискання лівою кнопкою миші по кнопці but (рядок 19). У першому випадку викликається функція color, у другому – outgo.

Функція color змінює колір фону (bg) кадру (fra) за допомогою методу configure, який призначений для зміни значення властивостей віджетів у процесі скрипту. Як значення опції bg підставляється перший елемент списку. Потім у списку два елементи змінюються місцями, щоб при наступному натисканні Enter колір кадру знову змінився.

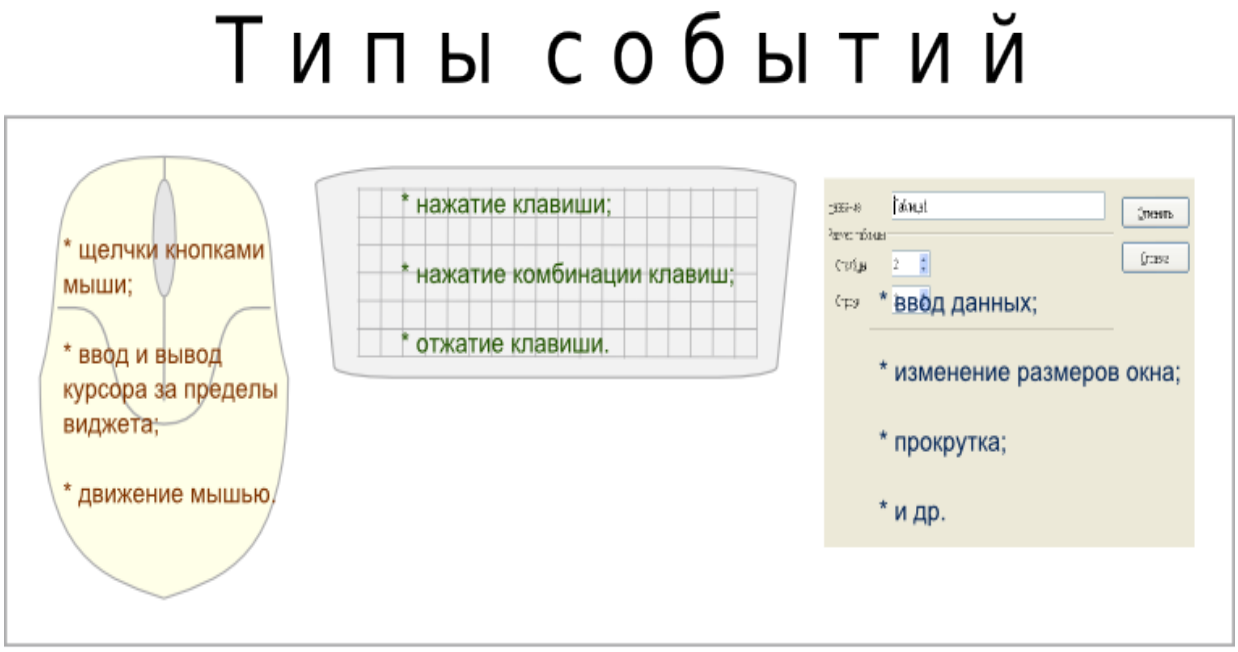
У функції outgo викликається метод destroy по відношенню до головного вікна. Цей метод призначений для «руйнування» віджету (вікно закриється).

Програмування подій у Tkinter

Зазвичай, щоб графічний додаток щось зробило, має статися якесь подія, т. е. вплив на GUI з-за.

Типи подій

Можна виділити три основні типи подій: миші, натискання клавіш на клавіатурі, а також події, що виникають в результаті зміни інших графічних об'єктів.



Спосіб запису

При виклику методу bind подія передається як перший аргумент.

Назва події полягає в лапки, а також знаки < і >. Подія описується за допомогою зарезервованих послідовностей ключових слів.

Події, що виробляються мишею

- **<Button-1>**- клацання лівою кнопкою миші
- **<Button-2>**- клацання середньою кнопкою миші
- **<Button-3>**- клацання правою кнопкою миші
- **<Double-Button-1>**- подвійний клік лівою кнопкою миші
- **<Motion>**- рух миші

і т.д.

Приклад:

```
з tkinter import *
defb1(event):
     root.title("Ліва кнопка миші")
defb3(event):
     root.title("Права кнопка миші")
defmove(event):
     root.title("Рух мишею")

root = Tk()
root.minsize (width = 500, height = 400)

root.bind('<Button-1>',b1)
root.bind('<Button-3>',b3)
root.bind('<Motion>',move)

root.mainloop()
```

У цій програмі змінюється напис у заголовку головного вікна залежно від того, що рухається миша, клацають лівою або правою кнопкою миші.

Подія (Event) – це один із об'єктів tkinter. Події мають атрибути, як і багато інших об'єктів. У прикладі функції move витягуються значення атрибутів x і y об'єкта event, в яких зберігаються координати розташування курсору миші в межах віджету, по відношенню до якого була згенерована подія. У разі віджетом є головне вікно, а подією – <Motion>, т. е. переміщення миші.

У програмі нижче виводиться інформація про примірник Event та деякі його властивості. Усі атрибути можна переглянути за допомогою команди dir(event). У різних подій вони ті самі, змінюються лише значення. Для тих чи інших подій частина атрибутів немає сенсу, такі властивості мають значення за умовчанням.

У прикладі хоча обробляється подія натискання клавіші клавіатури, поля x, y, x_root, y_root зберігаються координати положення на екрані курсора миші.

```
fromtkinter import *
```

```
defevent_info(event):
    print(type(event))
    print(event)
    print(event.time)
    print(event.x_root)
    print(event.y_root)


root=Tk()
root.bind('a',event_info)
root.mainloop()
```

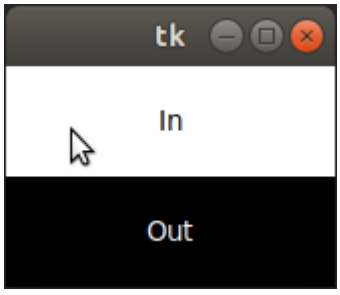
Приклад виконання програми:

```
<class 'tkinter.Event'>
<KeyPress event state=Mod2 keysym=a keycode=38
                              char='a' x=9 y=7>
8379853
37
92
```

Для подій з клавіатури можна записати буквені клавіші без кутових дужок (наприклад, 'a').

Для неалфавітних кнопок існують спеціальні зарезервовані слова. Наприклад, <Return> - натискання клавіші Enter, <space> - пробіл. (Зауважимо, що є подія <Enter>, яка не має відношення до натискання клавіші Enter, а відбувається, коли курсор заходить у межі віджету.)

Розглянемо програму:




```
fromtkinter import *


defenter_leave(event):
    if event.type == '7':
        event.widget['text'] ='In'
    elif event.type =='8':
        event.widget['text'] ='Out'


root=Tk()
```

```
lab1=Label(width=20,height=3, bg='white')
lab1.pack()
lab1.bind('<Enter>',enter_leave)
lab1.bind('<Leave>',enter_leave)

lab2=Label(width=20,height=3,bg='black',
             fg='white')
lab2.pack()
lab2.bind('<Enter>',enter_leave)
lab2.bind('<Leave>',enter_leave)
root.mainloop()
```

У ній дві мітки використовують одну й ту саму функцію, і кожна мітка використовує цю функцію для обробки двох різних подій: введення курсору в межі віджету та виведення у межі.

Функція, залежно від того, стосовно якого віджету було зафіксовано подію, змінює властивості тільки цього віджету. Як змінює, залежить від події, що сталася.

Властивість event.widget містить посилання на віджет, що згенерував подію.

Властивість event.type описує, що це була подія.

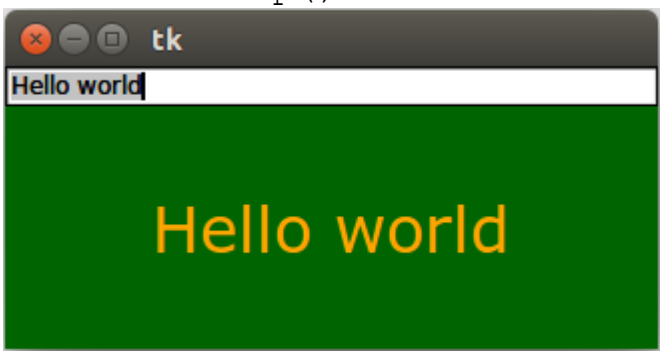
Кожна подія має ім'я та номер. За допомогою виразу print(repr(event.type)) можна переглянути його повний опис.

При цьому на одних платформах str(event.type) повертає ім'я події (наприклад, Enter), на інших – рядкове представлення номера події (наприклад, 7).

Повернімося до подій клавіатури. Поєднання клавіш пишуться через тире. У разі використання так званого модифікатора він вказується першим, деталі на третьому місці. Наприклад, <Shift-Up> - одночасне натискання клавіш Shift та стрілки вгору, <Control-B1-Motion> – рух мишею із затиснутою лівою кнопкою та клавішею Ctrl.

```
fromtkinter import *


defexit_win(event):
    root.destroy()


defto_label(event):
    t=ent.get()
    lbl.configure(text=t)


defselect_all(event):
```

```
    def select_all2(widget):
        widget.selection_range(0,END)
        widget.icursor(END)   курсор в кінець

    root.after(10,select_all2,event.widget)


root=Tk()

ent=Entry(width=40)
ent.focus_set()
ent.pack()
lbl=Label(height=3,fg='orange',
            bg='darkgreen',font="Verdana 24")
lbl.pack(fill=X)

ent.bind('<Return>',to_label)
ent.bind('<Control-a>',select_all)
root.bind('<Control-q>',exit_win)

root.mainloop()
```



Тут поєднання клавіш Ctrl+a виділяє текст у полі. Без root.after() виділення не працює. Метод after виконує функцію, вказану у другому аргументі, через проміжок часу, вказаний у першому аргументі. У третьому аргументі передається значення атрибута widget об'єкта event. У цьому випадку їм буде поле ent. Саме воно буде передано як аргумент у функцію select_all2 та присвоєно параметру widget.

Події, що виробляються за допомогою клавіатури

- Літерні клавіші можна записувати без кутових дужок (наприклад, 'L').
- Для неалфавітних клавіш існують спеціальні зарезервовані слова * <Return> - натискання клавіші Enter; * <space> - пробіл; * і т.д.

- Поєднання клавіш пишуться через тире. Наприклад: * <Control-Shift> - одночасне натискання клавіш Ctrl та Shift.

Прикладник:

```
з tkinter import *

defexit_(event):
     root.destroy()
defcaption(event):
     t = ent.get()
     lbl.configure(text = t)

root = Tk()

ent = Entry (root, width = 40)
lbl = Label(root, width = 80)

ent.pack()
lbl.pack()

ent.bind('<Return>',caption)
root.bind('<Control-z>',exit_)

root.mainloop()
```

При натисканні клавіші Enter в межах текстового рядка (ent) викликається функція caption, яка поміщає символи з текстового рядка (ent) до мітки (lbl). Натискання комбінації клавіш Ctrl+z призводить до закриття головного вікна.

XXIV. Змінні Tkinter.

Бібліотека Tkinter містить спеціальні класи, об'єкти яких виконують роль змінних для зберігання значень про стан різних віджетів. Зміна значення такої змінної веде до зміни властивості віджету, і навпаки: зміна властивості віджета змінює значення асоційованої змінної.

Існує кілька класів Tkinter, призначених для обробки даних різних типів.

1. **StringVar**() - для рядків;
2. **IntVar**() - цілих чисел;
3. **DoubleVar**() - дробових чисел;
4. **BooleanVar**() - для обробки булевих значень (true та false).

приклад 1.

Раніше ми вже використовували змінну-об'єкт типу IntVar() при створенні групи радіокнопок:

```
var=IntVar()
var.set(1)
rad0 =
Radiobutton(root,text="Перша",variable=var,value=0)

rad1 =
Radiobutton(root,text="Друга",variable=var,value=1)

rad2 =
Radiobutton(root,text="Третя",variable=var,value=2)
```

Тут створюється об'єкт класу IntVar та зв'язується зі змінною var. За допомогою методу set встановлюється початкове значення 1. Три радіокнопки відносяться до однієї групи: про це свідчить однакове значення опції (властивості) variable. Variable призначена для зв'язування змінної Tkinter із радіокнопкою. Опція value визначає значення, яке буде передано змінною, якщо ця кнопка буде у стані "увімкнено". Якщо у процесі виконання скрипта значення змінної var буде змінено, це позначиться групі кнопок. Наприклад, це робиться в другому рядку коду: увімкнена кнопка rad1.

Якщо метод set дозволяє встановлювати значення змінних, то метод get, навпаки, дозволяє отримувати (пізнавати) значення подальшого їх використання.

```
def display(event):
     v = var.get()
     if v == 0:
          print ("Увімкнена перша кнопка")
     elif v == 1:
          print ("Включена друга кнопка")
     elif v == 2:
          print ("Увімкнена третя кнопка")

but = Button(root,text="Отримати значення")
but.bind('<Button-1>',display)
```

При виклику функції display в змінну v “записується“ значення, пов'язане з змінною var. Щоб отримати значення змінної var, використовується метод get (другий рядок коду).

приклад 2.

Дещо складніше справа з прапорцями. Оскільки стани прапорців незалежні один одного, то для кожного має бути введена власна асоційована змінна-об'єкт.

```
з tkinter import *

root = Tk()

var0=StringVar() # значення кожного прапорця ...
```

```
var1=StringVar() # ... зберігається у власній змінній
var2=StringVar()
# якщо прапорець встановлений, то асоційовану
змінну ...
# ... (var0, var1 або var2) заноситься значення
onvalue, ...
# ...якщо прапорець знято, то - offvalue.
ch0 = Checkbutton(root,text="Коло",variable=var0,
          onvalue="circle",offvalue="-")
ch1 = Checkbutton(root,text="Квадрат",variable=var1,
          onvalue="square",offvalue="-")
ch2 = Checkbutton(root,text="Трикутник",variable=var2,
          onvalue="triangle",offvalue="-")

lis = Listbox(root,height=3)
defresult(event):
     v0 = var0.get()
     v1 = var1.get()
     v2 = var2.get()
     l = [v0,v1,v2] # значення змінних заносяться до
списку
     lis.delete(0,2) # попередній вміст видаляється з
Listbox
     for v in l: # вміст списку l послідовно
          lis.insert(END,v) # ...вставляється в Listbox

but = Button(root,text="Отримати значення")
but.bind('<Button-1>',result)

ch0.deselect()# "за замовчуванням" прапорці зняті
ch1.deselect()
ch2.deselect()

ch0.pack()
ch1.pack()
ch2.pack()
but.pack()
lis.pack()

root.mainloop()
```

приклад 3.

Крім властивості (опції) variable, що зв'язує віджет зі змінною-об'єктом Tkinter (IntVar, StringVar та ін), у багатьох віджет існує опція textvariable, яка визначає текст-вміст або текст-напис віджету. Незважаючи на те, що «текстова властивість» може бути встановлена для віджету і змінена в

процесі виконання коду без використання асоційованих змінних, іноді такий спосіб зміни виявляється зручнішим.

```
з tkinter import *
root = Tk()
v = StringVar()
ent1 = Entry (root, textvariable = v, bg = "black", fg
= "white")

ent2 = Entry(root, textvariable = v)
ent1.pack()
ent2.pack()
root.mainloop()
```

Тут вміст одного текстового поля негайно відображається в іншому, т.к. обидва поля прив'язані до однієї і тієї ж змінної v.

XXV. Об'єкт Меню в GUI.

Що таке меню

Меню — це об'єкт, який присутній у багатьох програмах користувача. Знаходиться воно під рядком заголовка і являє собою списки, що випадають під словами; кожен такий список може містити інший вкладений список. Кожен пункт списку є командою, яка запускає будь-яку дію або відкриває діалогове вікно.

Створення меню в Tkinter

```
fromtkinter import *
root = Tk()

m = Menu(root) # створюється об'єкт Меню на головному
вікні
root.config(menu=m) #вікно конфігурується із
зазначенням меню для нього


fm = Menu(m) # створюється пункт меню з розміщенням на
основному меню (m)

m.add_cascade(label="File",menu=fm) #пункту
розташовується на основному меню (m)


fm.add_command(label="Open...") #формується список
команд пункту меню

fm.add_command(label="New")
fm.add_command(label="Save...")
```

```
fm.add_command(label="Exit")

hm = Menu(m) #другий пункт меню
m.add_cascade(label="Help",menu=hm)
hm.add_command(label="Help")
hm.add_command(label="About")

root.mainloop()
```

Метод add_cascade додає новий пункт до меню, який вказується як значення опції menu.

Метод add_command додає нову команду до пункту меню. Одна з опцій даного методу (у прикладі вище її поки що немає) - command - пов'язує цю команду з функцією-обробником.

Можна створити вкладене меню. Для цього створюється ще одне меню та за допомогою add_cascade прив'язати до батьківського пункту.

```
nfm = Menu(fm)
fm.add_cascade(label="Import",menu=nfm)
nfm.add_command(label="Image")
nfm.add_command(label="Text")
```

Прив'язка функцій до меню

Кожна команда меню зазвичай має бути пов'язана зі своєю функцією, яка виконує ті чи інші дії (вирази). Зв'язок відбувається з допомогою опції command методу add_command. Функція оброблювач до цього має бути визначена.

Для прикладу вище наведено виправлені рядки додавання команд “About”, “New” і “Exit”, а також функції, що викликаються, коли користувач клацає лівою кнопкою миші по відповідним пунктам підменю.

```
defnew_win():
     win = Toplevel(root)

defclose_win():
     root.destroy()

defabout():
     win = Toplevel(root)
     lab = Label(win,text="Це просто програма-тест \n
меню в Tkinter")
     lab.pack()
….
fm.add_command(label="New",command=new_win)
….
fm.add_command(label="Exit",command=close_win)

….
hm.add_command(label="About",command=about)
```

Вправа – приклад

Напишемо програму з меню, що містить два пункти: Color і Size. Пункт Color повинен містити три команди (Red, Green та Blue), що змінюють колір рамки на головному вікні. Пункт Size повинен містити дві команди (500x500 та 700x400), що змінюють розмір рамки.

Зразкове рішення:

```
fromtkinter import *
root = Tk()

defcolorR():
     fra.config(bg="Red")
defcolorG():
     fra.config(bg="Green")
defcolorB():
     fra.config(bg="Blue")

defsquare():
     fra.config(width=500)
     fra.config(height=500)
defrectangle():
     fra.config(width=700)
     fra.config(height=400)

fra = Frame (root, width = 300, height = 100, bg =
"Black")
fra.pack()

m = Menu(root)
root.config(menu=m)

cm = Menu(m)
m.add_cascade(label="Color",menu=cm)
cm.add_command(label="Red",command=colorR)
cm.add_command(label="Green",command=colorG)
cm.add_command(label="Blue",command=colorB)

sm = Menu(m)
m.add_cascade(label="Size",menu=sm)
sm.add_command(label="500x500",command=square)
sm.add_command(label="700x400",command=rectangle)

root.mainloop()
```

XXVI. Діалогові вікна в Tkinter

Діалогові вікна, як елементи графічного інтерфейсу, призначені для виведення повідомлень користувачеві, отримання від нього будь-якої інформації, а також управління.

Діалогові вікна дуже різноманітні. Тут буде розглянуто лише кілька.

Розглянемо, як запрограмувати за допомогою Tkinter виклик діалогових вікон відкриття та збереження файлів та роботу з ними. При цьому потрібно імпортувати "підмодуль" Tkinter - tkinter.filedialog, в якому описані класи для вікон даного типу.

```
з tkinter import *
з tkinter.filedialog import *

root = Tk()
op = askopenfilename()
sa = asksaveasfilename()

root.mainloop()
```

Тут створюються два об'єкти (op та sa): один викликає діалогове вікно "Відкрити", а інший "Зберегти як...". При виконанні скрипта вони один за одним виводяться на екран після появи головного вікна. Якщо не створити root, то воно все одно з'явиться на екрані, проте при спробі його закриття в кінці виникне помилка.

Давайте розмістимо багаторядкове текстове поле на головному вікні і надалі спробуємо туди завантажувати вміст невеликих текстових файлів. Оскільки вікно збереження файлу нам поки що не потрібно, то закоментуємо цей рядок коду або видалимо. В результаті має вийти приблизно так:

```
з tkinter import *
з tkinter.filedialog import *

root = Tk()
txt = Text (root, width = 40, height = 15, font = "12")
txt.pack()

op = askopenfilename()

root.mainloop()
```

Під час запуску скрипта з'являється вікно з текстовим полем і відразу діалогове вікно "Відкрити". Однак, якщо ми спробуємо відкрити текстовий файл, то в кращому випадку нічого не станеться. Як зв'язати вміст текстового файлу з текстовим полем через діалог "Відкрити"?

Якщо просто вставити вміст змінної op в текстове поле:

```
txt.insert(END,op)
```

Після запуску скрипта та спроби відкриття файлу в текстовому полі надається адреса файлу. Значить, вміст файлу треба прочитати якимось методом (функцією).

Метод input модуля fileinput може приймати як аргумент адресу файлу, читати його вміст, формуючи список рядків. Далі за допомогою циклу for

можна витягувати рядки послідовно та поміщати їх, наприклад, у текстове поле.

```
.....
importfileinput
.....
fori in fileinput.input(op):
     txt.insert(END,i)
.....
```

Зверніть увагу на те, як відбувається звернення до функції input модуля fileinput та його імпорту. Справа в тому, що в Python вже вбудована своя функція input (її призначення абсолютно інше) і, щоб уникнути "конфлікту", потрібно чітко вказати, яку саме функцію ми маємо на увазі. Тому варіант імпорту 'from fileinput import input' тут не підходить.

Вікно "Відкрити" запускається одразу при виконанні скрипта. Насправді так не має бути. Необхідно пов'язати запуск вікна з якоюсь подією. Нехай це буде клацання на пункті меню.

```
fromtkinter import *
fromtkinter.filedialog import *
importfileinput

def_open():
     op = askopenfilename()
     for l in fileinput.input(op):
          txt.insert(END,l)

root = Tk()

m = Menu(root)
root.config(menu=m)

fm = Menu(m)
m.add_cascade(label="File",menu=fm)
fm.add_command(label="Open...",command=_open)

txt = Text (root, width = 40, height = 15, font = "12")
txt.pack()

root.mainloop()
```

Тепер спробуємо зберегти текст, набраний у текстовому полі. Додамо в код пункт меню та наступну функцію:

```
def_save():
     sa = asksaveasfilename()
     letter = txt.get(1.0,END)
     f = open(sa, "w")
     f.write(letter)
     f.close()
```

У змінній sa зберігається адреса файлу, куди буде записуватися. У змінній letter – текст, "отриманий" із текстового поля. Потім файл відкривається для запису, в нього записується вміст змінної letter, і файл закривається (про всяк випадок).

Ще одна група діалогових вікон описана у модулі tkinter.messagebox. Це досить прості діалогові вікна для виведення повідомлень, попереджень, отримання від користувача відповіді так чи ні тощо.

Доповнимо нашу програму пунктом Exit у підменю File і пунктом About program у підменю Help.

```
fromtkinter.messagebox import *
….
defclose_win():
     if askyesno("Exit", "Do you want to quit?"):
          root.destroy()

defabout():
     showinfo("Editor", "This is text editor.\n
                                        (test version)")
...
fm.add_command(label="Exit",command=close_win)
....
hm = Menu(m)
m.add_cascade(label="Help",menu=hm)
hm.add_command(label="About",command=about)
...
```

У функції про відбувається виклик вікна showinfo, що дозволяє виводити повідомлення для користувача з кнопкою OK. Перший аргумент - це те, що виведеться в заголовку вікна, а другий - те, що буде у тілі повідомлення. У функції close_win викликається вікно askyesno, яке дозволяє отримати від користувача дві відповіді (true і false). У разі при позитивному відповіді спрацює гілка if головне вікно буде закрито. У разі натискання користувачем кнопки "No" вікно просто закриється (хоча можна було запрограмувати у гілці else будь-яку дію).

XXVII. Геометричні примітиви графічного елемента Canvas (полотно)

Canvas (полотно) – це досить складний об'єкт бібліотеки tkinter. Він дозволяє розташовувати на собі інші об'єкти. Це може бути як геометричні фігури, візерунки, вставлені зображення, і інші виджети (наприклад, мітки, кнопки, текстові поля). І це ще не все. Відображені на полотні об'єкти можна змінювати та переміщувати (за бажанням) у процесі виконання скрипту. Враховуючи все це, canvas знаходить широке застосування при створенні

GUI-додатків з використанням tkinter (створення малюнків, оформлення інших віджетів, реалізація функцій графічних редакторів, програмована анімація та ін.).

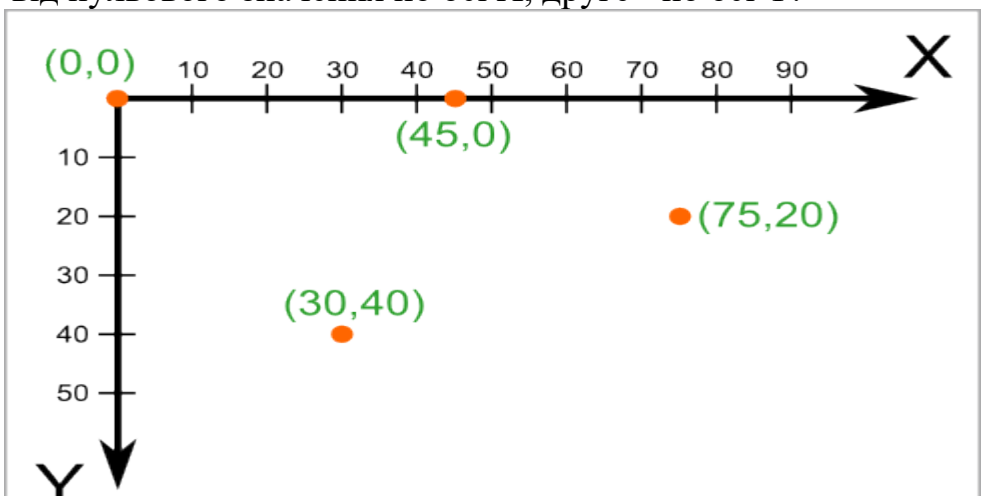
У цьому уроці буде розглянуто створення на полотні графічних примітивів (лінії, прямокутника, багатокутника, дуги (сектору), еліпса) та тексту.

Для того, щоб створити об'єкт-полотно, необхідно викликати відповідний клас модуля tkinter і встановити деякі значення властивостей (опцій). Наприклад:

```
canv = Canvas(root,width=500,height=500,bg="lightblue",
                                                cursor =
"pencil")
```

Далі за допомогою будь-якого менеджера з геометрії розмістити на головному вікні.

Перед тим, як створювати геометричні фігури на полотні, слід розібратися з координатами та одиницями вимірювання відстані. Нульова точка (0,0) для об'єкта Canvas розташована у верхньому лівому кутку. Одиниці виміру пікселі (точки екрану). Для «орієнтації у просторі» об'єкта Canvas розгляньте малюнок нижче. У будь-якій точці перше число - це відстань від нульового значення по осі X, друге - по осі Y.



Щоб намалювати лінію на полотні слід до об'єкта (у нашому випадку canv) застосувати метод create_line.

```
canv.create_line(200,50,300,50,width=3,fill="blue")
canv.create_line(0,0,100,100,width=2,arrow=LAST)
```

Чотири числа - це пари координат початку та кінця лінії, тобто в прикладі перша лінія починається з точки (200,50), а закінчується в точці (300,50). Друга лінія починається в точці (0,0), закінчується - (100,100).

Властивість fill дозволяє задати колір лінії відмінний від чорного, а arrow – встановити стрілку (наприкінці, на початку або на обох кінцях лінії).

Метод create_rectangle створює прямокутник. Аналогічно лінії у дужках першими аргументами прописуються чотири числа. Перші дві координати позначають верхній лівий кут прямокутника, другі правий нижній. У прикладі нижче використовується дещо інший підхід. Він може бути

корисним, якщо початкові координати об'єкта можуть змінюватися, яке розмір суворо регламентований.

```
x = 75
y = 110
canv.create_rectangle(x,y,x+80,y+50,fill="white",
outline="blue")
```

Опція outline визначає колір межі прямокутника.

Щоб створити довільний багатокутник, потрібно задати пари координат кожної його точки.

```
canv.create_polygon([250,100],[200,150],[300,150],
fill="yellow")
```

Квадратні дужки при заданні координат використовуються для зручності читання (їх можна не використовувати). Властивість smooth задає згладжування.

```
canv.create_polygon([250,100],[200,150],[300,150],
fill="yellow")
canv.create_polygon([300,80],[400,80],
[450,75],[450,200], [300,180],[330,160],
outline="white",smooth=1)
```

При створенні еліпса задаються координати гіпотетичного прямокутника, що описує цей еліпс.

```
canv.create_oval([20,200],[150,300],fill="gray50")
```

Більш складні розуміння фігури виходять під час використання методу create_arc. Залежно від значення опції style можна отримати сектор (за умовчанням), сегмент (CHORD) або дугу (ARC). Координати, як і раніше, задають прямокутник, в який вписано коло, з якого «вирізають» сектор, сегмент або дугу. Від опцій start та extent залежить кут фігури.

```
canv.create_arc([160,230],[230,330],start=0,extent=140,
fill="lightgreen")
canv.create_arc([250,230],[320,330],start=0,extent=140,
style=CHORD,fill="green")

canv.create_arc([340,230],[410,330],start=0,extent=140,
                style=ARC,outline="darkgreen",width=2)
```

Останній метод об'єкта canvas, який буде розглянутий у цьому уроці - це метод, що створює текстовий напис.

```
canv.create_text(20,330, text="Досліди з графічними
примітивами\nна полотні", font="Verdana
12",anchor="w",justify=CENTER, fill="red")
```

Труднощі тут можуть виникнути з розумінням опції anchor (якір). За замовчуванням у заданій координаті розміщується центр текстового напису. Щоб змінити це та, наприклад, розмістити за вказаною координатою ліву

межу тексту, використовується якір зі значенням w (від англ. west – захід). Інші значення: n, ne, e, se, s, sw, w, nw.

Якщо букв, що задають бік прив'язки дві, то друга визначає вертикальну прив'язку (вгору чи вниз «піде» текст від координати). Властивість justify визначає лише вирівнювання тексту щодо себе самого.

Наприкінці слід зазначити, що часто потрібно «намалювати» на полотні будь-які елементи, що повторюються. Щоб не завантажувати код, використовують цикли. Наприклад, так:

```
x=10
while x < 450:
      canv.create_rectangle(x,400,x+50,450)
      x = x + 60
```

Якщо ви напишіть код наведений у даному уроці (попередньо здійснивши імпорт модуля Tkinter і створення головного вікна, а також не забувши розташувати на вікні полотно, і в кінці "зробити" mainloop), то при його виконанні побачите таку картину:

Canvas (полотно) - методи, ідентифікатори та теги

Раніше були розглянуті методи об'єкта canvas, що формують на ньому геометричні примітиви та текст. Однак це лише частина методів полотна. В іншу умовну групу можна виділити методи, що змінюють властивості існуючих об'єктів полотна (наприклад, геометричних фігур). І тут постає питання: як звертатися до вже створених фігур? Адже якщо під час створення було прописано щось на кшталт canvas.create_oval(30,10,130,80) і таких овалів, квадратів та ін. на полотні дуже багато, то як до них звертатися?

Для вирішення цієї проблеми в tkinter для об'єктів полотна можна використовувати ідентифікатори та теги, які потім передаються іншим методам. Будь-який об'єкт може бути як ідентифікатор, так і тег. Використання ідентифікаторів та тегів трохи відрізняється.

Розглянемо кілька методів зміни вже існуючих об'єктів з використанням ідентифікаторів. Для початку створимо полотно та три об'єкти на ньому. При створенні об'єкти "повертають" свої ідентифікатори, які можна пов'язати зі змінними (oval, rect та trial у прикладі нижче) і потім використовувати їх для звернення до конкретного об'єкта.

```
c = Canvas(width=460,height=460,bg='grey80')
c.pack()
oval = c.create_oval(30,10,130,80)
rect = c.create_rectangle(180,10,280,80)
trian = c.create_polygon(330,80,380,10,430,80,
fill='grey80', outline="black")
```

Якщо ви виконаєте цей скрипт, побачите на полотні три фігури: овал, прямокутник і трикутник.

Далі можна використовувати методи-модифікатори вказуючи в якості першого аргументу ідентифікатор об'єкта. Метод move переміщає об'єкт на осі X і Y на відстань зазначене як другий і третій аргументів. Слід розуміти, що це координати, а зміщення, т. е. у прикладі нижче прямокутник опуститься вниз на 150 пікселів. Метод itemconfig змінює зазначені властивості об'єктів, coords змінює координати (їм можна змінювати і розмір об'єкта).

```
c.move(rect,0,150)
c.itemconfig(trian,outline="red",width=3)
c.coords(oval,300,200,450,450)
```

Якщо запустити скрипт, що містить дві наведені частини коду (один за одним), то ми відразу побачимо картину, що вже змінилася, на полотні: прямокутник опуститься, трикутник придбає червоний контур, а еліпс зміститься і сильно збільшиться в розмірах. Зазвичай у програмах зміни повинні наступати при якомусь зовнішньому впливі. Нехай по клацанню лівою кнопкою миші прямокутник пересувається на два пікселі вниз (він буде робити це при кожному клацанні мишею):

```
def mooove(event):
    c.move(rect,0,2)
...
c.bind('<Button-1>',mooove)
```

Тепер розглянемо, як працюють теги. На відміну від ідентифікаторів, які є унікальними для кожного об'єкта, той самий тег може присвоюватися різним об'єктам. Подальше звернення до такого тегу дозволить змінити всі об'єкти, де він був вказаний. У прикладі нижче еліпс та лінія містять один і той же тег, а функція color змінює колір усіх об'єктів з тегом group1. Зверніть увагу, що на відміну від імені ідентифікатора (змінна), ім'я тега полягає в лапки (рядкове значення).

```
oval = c.create_oval(30,10,130,80,tag="group1")
c.create_line(10,100,450,100,tag="group1")
…
```

```
defcolor(event):
     c.itemconfig('group1',fill="red",width=3)
…
c.bind('<Button-3>',color)
```

Ще один метод, який варто розглянути, це delete, який видаляє об'єкт за вказаним ідентифікатором або тегом. У tinter є зарезервовані теги: наприклад, all позначає всі об'єкти полотна. Так, у прикладі нижче функція clean просто очищає полотно.

```
def clean(event):
     c.delete('all')
...
c.bind('<Button-2>',clean)
```

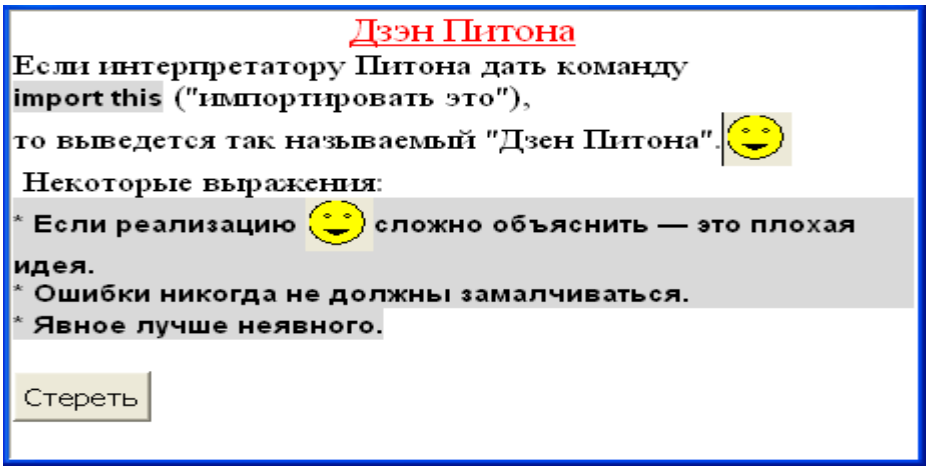
Метод tag_bind дозволяє прив'язати подію (наприклад, клацання кнопкою миші) до певного об'єкта. Таким чином, можна реалізувати звернення до різних областей полотна за допомогою однієї і тієї ж події. Приклад нижче це наочно ілюструє: зміни на полотні залежать від того, де зроблено клацання мишею.

```
fromtkinter import *

c = Canvas(width=460,height=100,bg='grey80')
c.pack()

oval = c.create_oval(30,10,130,80,fill="orange")
c.create_rectangle(180,10,280,80,tag="rect",
fill="lightgreen")

trian = c.create_polygon(330,80,380,10,430,80,
fill='white',outline="black")


defoval_func(event):
     c.delete(oval)
     c.create_text(30,10, text="Тут було
коло",anchor="w")
defrect_func(event):
     c.delete("rect")
     c.create_text(180,10, text="Тут
був\nпрямокутник",anchor="nw")
deftriangle(event):
     c.create_polygon(350,70,380,20,410,70,
fill='yellow',outline="black")

c.tag_bind(oval,'<Button-1>',oval_func)
c.tag_bind("rect",'<Button-1>',rect_func)
c.tag_bind(trian,'<Button-1>',triangle)
```

```
mainloop()
```

Особливості роботи із віджетом Text модуля Tkinter.

Графічний елемент Text надає великі можливості для роботи з текстовою інформацією. Крім різноманітних операцій з текстом та його форматуванням в екземпляр об'єкта Text можна вставляти інші віджети (слід зазначити, що така ж можливість існує і для Canvas). У цьому уроці розглядаються лише деякі можливості віджету Text на прикладі створення вікна з текстовим полем, що містить форматований текст, кнопку та можливість додавання екземплярів полотна.



1. Для початку створимо текстове поле, встановивши при цьому деякі з його властивостей:

```
#Текстове поле та його початкові налаштування
tx = Text
(font=('times',12), Width =50,height =15,wrap=WORD)
tx.pack(expand=YES,fill=BOTH)
```

2. Тепер допустимо нам потрібно додати якийсь текст. Зробити це можна за допомогою методу insert, передавши йому два обов'язкові аргументи: місце, куди вставити, та об'єкт, який слід вставити. Об'єктом може бути рядок, змінний, що посилається на рядок або на будь-який інший об'єкт. Місце вставки може вказуватись декількома способами. Один із них — це індекси. Вони записуються як 'xy', де x – це рядок, а y – стовпець. У цьому нумерація рядків починається з одиниці, а стовпців з нуля. Наприклад, перший символ у першому рядку має індекс '1.0', а десятий символ у п'ятому рядку - '5.9'.

```
tx.insert(1.0,'Дзен Пітона\n\
Якщо інтерпретатору Пітона дати команду\n\
importthis ("імпортувати це"),\n\
       то виведеться так званий "Дзен Пітона".\n Деякі вирази:\n\
       * Якщо реалізацію складно пояснити - це погана ідея. Помилки ніколи не повинні замовчуватися. * Явне краще неявного.
```

Комбінація символів '\n' створює новий рядок (тобто при інтерпретації наступний текст розпочнеться з нового рядка). Одиночний символ '\' ніяк не впливає на відображення тексту під час виконання коду, його слід вставляти при перенесенні тексту під час написання скрипта.

Якщо вміст текстового поля немає взагалі, то єдиний доступний індекс - '1.0'. У заповненому текстовому полі можна вставляти в будь-яке місце (де є вміст).

Якщо виконати скрипт, що містить тільки цей код (+ імпорт модуля Tkinter, + створення головного вікна, + mainloop() наприкінці), ми побачимо текстове поле з вісьмома рядками тексту. Текст не оформлено.

3. Відформатуємо різні області тексту по-різному. Для цього спочатку задамо теги для потрібних нам областей, а потім для кожного тега встановимо налаштування шрифту та ін.

```
#установка тегів для областей тексту
tx.tag_add('title','1.0','1.end')
tx.tag_add('special','6.0','8.end')
tx.tag_add('special','3.0','3.11')

#конфігурування тегів
tx.tag_config('title',foreground='red',
          font=('times',14,'underline'),justify=CENTER)
tx.tag_config('special',background='grey85',
font=('Dejavu',10,'bold'))
```

Додавання тега здійснюється за допомогою методу tag_add. Перший атрибут - ім'я тега (довільне), далі за допомогою індексів вказується до якої області текстового поля він прикріплюється (початковий символ і кінцевий). Варіант запису як '1.end' говорить про те, що потрібно взяти текст до кінця вказаного рядка. Різні області тексту можуть бути позначені однаковим тегом.

Метод tag_config застосовує ті чи інші властивості до тега, вказаного як перший аргумент.

4. У багаторядкове текстове поле можна додавати не лише текст, а й інші об'єкти. Наприклад, вставимо в поле кнопку (та й функцію заразом).

```
def erase():
     tx.delete('1.0',END)
...
#додавання кнопки
bt = Button(tx,text='Стерти',command=erase)
tx.window_create(END,window=bt)
```

Кнопка – це віджет. Віджети додаються в текстове поле за допомогою методу window_create, де як перша опція вказується місце додавання, а другий (window) - як значення присвоюється змінна, пов'язана з об'єктом.

При натисканні ЛКМ (лівою кнопкою миші) по кнопці буде викликатися функція erase, в якій за допомогою методу delete видаляється весь вміст поля (від 1.0 до END).

5. А ось цікавіший приклад додавання віджету в поле Text:

```
defsmiley(event):
      cv = Canvas (height = 30, width = 30)
      cv.create_oval(1,1,29,29,fill="yellow")
      cv.create_oval(9,10,12,12)
      cv.create_oval(19,10,22,12)
      cv.create_polygon(9,20,15,24,22,20)
      tx.window_create(CURRENT,window=cv)
…
#ЛКМ -> смайлик
tx.bind('<Button-1>',smiley)
```

Тут при натисканні ЛКМ у будь-якому місці текстового поля буде викликатися функція smiley. У тілі цієї функції створюється об'єкт полотна, який наприкінці з допомогою методу window_create додається об'єкт tx. Місце вставки вказано як CURRENT, тобто "поточне" - це там, де було зроблено клацання мишею.

XXVIII. Декілька прикладів

Гра «життя»

(за https://habrahabr.ru/company/mailru/blog/228379/)

```
з tinter import Tk, Canvas, Button, Frame, BOTH, NORMAL, HIDDEN
# функція отримує координати миші # в момент натискання або
пересування із затиснутою кнопкою
defdraw_a(e):
     ii = (ey-3) / cell_size
     jj = (ex-3) / cell_size
     ii = int (ii); jj = int (jj)
     print("e=",e, 'ii=',ii, 'jj=',jj)
     canvas.itemconfig(cell_matrix[addr(ii, jj)], state=NORMAL,
tags='vis')
```

ця функція перетворює двовимірну координату в простий адресу одномірного масиву

```
defaddr(ii,jj):
     if(ii < 0 or jj < 0 or ii >= field_height or jj >=
field_width):
          return int( len(cell_matrix)-1 )
     else:
          return int( ii * (win_width / cell_size) + jj )
```

```
# функція оновлення «картинки»
defrefresh():
     fori in range(field_height):
          forj in range(field_width):
               k= 0
              # вважаємо число сусідів
               fori_shift in range(-1, 2):
                    forj_shift in range(-1, 2):
                         if(canvas.gettags( cell_matrix[addr(i +
i_shift, j + j_shift)])[0] == 'vis' and (i_shift != 0 або
j_shift != 0)):



                              k += 1
               current_tag = canvas.gettags(cell_matrix[addr(i,
j)])[0]

               # в залежності від їх числа встановлюємо
стан клітини
               тут можна експериментувати з самими
"правилами" гри
               if(k == 3):
                    canvas.itemconfig(cell_matrix[addr(i, j)],
tags=(current_tag, 'to_vis'))

               if(k <= 1 або k > = 4):
                    canvas.itemconfig(cell_matrix[addr(i, j)],
tags=(current_tag, 'to_hid'))

               if(k == 2 and canvas.gettags(cell_matrix[ addr(i,
j)])[0] == 'vis'):


                   canvas.itemconfig(cell_matrix[addr(i, j)],
tags=(current_tag, 'to_vis'))


сам крок: оновлюємо стан і малюємо
defstep():
     refresh()
     repaint()

defclear():
     for i in range(field_height):
          for j in range(field_width):
               canvas.itemconfig(cell_matrix[addr(i, j)],
state=HIDDEN, tags=('hid','0'))


# перемальовуємо поле по буферу # згідно з другим тегом елемента
```

```
defrepaint():
     fori in range(field_height):
          for jinrange(field_width):
               if(canvas.gettags(cell_matrix[addr(i, j)])[1] ==
'to_hid'):

                    canvas.itemconfig(cell_matrix[addr(i, j)],
state=HIDDEN, tags=('hid','0'))

               if(canvas.gettags(cell_matrix[addr(i, j)])[1] ==
'to_vis'):

                    canvas.itemconfig(cell_matrix[addr(i, j)],
state=NORMAL, tags=('vis','0'))


# «створення та підтримка виконання» # автоматичного
режиму

defloop():
    ##label['text'] = str(n)
    step()
    if (flag_run):
            root.after(500, loop) # call loop() in 0.5 seconds

# «Виконання» автоматичного режиму
defrun():
        global flag_run, btn3
        #print('flag_run=',flag_run)
        if not flag_run:
                btn3.config(text="STOP")
                flag_run=True
                #print('stop')
                loop()
        else:
                btn3.config(text="RUN")
                flag_run=False
                #print('run')

flag_run=False # працює «автоматично» або по кроках

# створюємо саме вікно
root = Tk()
root.title('4 cuba')
# це ширина та висота вікна
win_width = 360 # 350
win_height = 380 # 370
# «будуємо» конфігураційний рядок – розмір вікна
config_string = "{0}x{1}".format(win_width, win_height + 32)
# Задаємо колір клітини
fill_color = "green"
```

```
методом geometry() задаємо розміри, можна написати і
просто рядок виду '360x380'
root.geometry(config_string)
це ширина самої клітини, з урахуванням «просвіту»
cell_size = 20
# створюється об'єкт типу Canvas, на якому буде
відбуватися безпосередньо малювання,

# він робиться дочірнім по відношенню до самого вікна
root
canvas = Canvas (root, height = win_height)
# пакуємо його, аналог show() в інших системах
canvas.pack(fill=BOTH)
# Визначаємо розміри поля в клітинах
field_height = int( win_height / cell_size )
field_width = int( win_width / cell_size )
# створюємо масив для клітин, він одномірний,
# але використовуючи допоміжну функцію addr –
# будемо перетворювати індекси 2-х мірний до 1-мірного
cell_matrix = []
тут у циклі створюємо екземпляри клітин
# і робимо їх прихованими
fori in range(field_height):
     forj in range(field_width):
               # тут створюємо екземпляр клітини # і
робимо його прихованими

          square = canvas.create_rectangle(2 + cell_size*j, 2 +
cell_size*i, cell_size + cell_size*j - 2, cell_size +
cell_size*i - 2, fill=fill_color)




          canvas.itemconfig(square, state=HIDDEN,
tags=('hid','0'))

           # «додаємо» до масиву
          cell_matrix.append(square)
# створимо фіктивний елемент, # він як би «повсюди»
поза полем

fict_square = canvas.create_rectangle(0,0,0,0, state=HIDDEN,
tags=('hid','0'))


cell_matrix.append(fict_square)
```

```
# задаємо «клітини» які знаходяться на полі # «за
замовчуванням»

canvas.itemconfig(cell_matrix[addr(8, 8)], state=NORMAL,
tags='vis')

canvas.itemconfig(cell_matrix[addr(10, 9)], state=NORMAL,
tags='vis')

canvas.itemconfig(cell_matrix[addr(9, 9)], state=NORMAL,
tags='vis')

canvas.itemconfig(cell_matrix[addr(9, 8)], state=NORMAL,
tags='vis')

canvas.itemconfig(cell_matrix[addr(9, 7)], state=NORMAL,
tags='vis')

canvas.itemconfig(cell_matrix[addr(10, 7)], state=NORMAL,
tags='vis')


# створюємо кадр для зберігання кнопок і # аналогічним
чином, як з Canvas,

# встановлюємо кнопки дочірні кадри

frame = Frame(root)
# виконати «крок»
btn1 = Button (frame, text = 'Step', command = step) # Eval
# очистити поле
btn2 = Button (frame, text = 'Clear', command = clear)
# Зміна режимів «автоматично» - «по кроках»
btn3 = Button (frame, text = 'Run', command = run)
# пакуємо кнопки

btn1.pack(side='left')
btn2.pack(side='right')
btn3.pack(side='right')
# пакуємо кадр
frame.pack(side='bottom')

# прив'язуємо події кліка і руху миші над canvas до
функції draw_a

canvas.bind('<B1-Motion>', draw_a)
canvas.bind('<ButtonPress>', draw_a)
через 500 мкс передати управління функції loop
root.after(500, loop)
```

стандартний цикл, що організовує події і загальну роботу віконного додатка

```
root.mainloop()
```

Різне

```
import sys
from math import *
з tkinter import *
з tkinter.filedialog import *
від tkinter.messagebox import *
#### http://younglinux.info/tkinter/canvas.php

deff1():
    print('f1 - run.........')
defb1(event):
     root.title("Ліва кнопка миші")
defb3(event):
     root.title("Права кнопка миші")
defmove(event):
     root.title("Рух мишею")
defres(event):
    root.title("зміна розміру вікна")
    print("зміна розміру вікна")
    xm, ym = xymax(root.geometry())
    print("--------xm---ym----------> ", xm, " ", ym)
    print("---------------------> ",canv. .geometry() )
    canv.config(width = xm, height = ym)

defxymax(str):
    """ отримаємо новий розмір вікна """
    print("===> ", str )
    k1=str.index('x')
    xmax1=str[:k1]
    k2=str.index('+',k1+1)
    ymax1=str[k1+1:k2]
    xmax = int (xmax1)
    ymax=int(ymax1)
    print(xmax,'',ymax)
    return xmax,ymax

defbutton_clicked():
    sss=root.geometry()
    print ("Клік!")
    print('root.maxsize()=<', root.maxsize(),'>' )
    print('canv.height()=<', canv.height(),'>' )
    print('canv.size()=<', canv.size(),'>' )
    print(root.geometry() )
    print('---canv.winfo_height()=',canv.winfo_height() )
     print('---canv.winfo_width() =',canv.winfo_width() )
     return
defwindow_deleted():
```

```
    print ('Вікно закрите')
    #canv.delete('all')
    if askyesno("Завершення програми", "Справді Ви хочете
завершити програму?"):
            #self.save_file()
            root.destroy()
    else:
            pass
   #root.destroy()
   # root.quit() # явна вказівка на вихід із програми
   # exit(0)
f = input('f(x):')

root = Tk()

f_1=Frame(root,bg="Yellow",)
f_2=Frame(root,bg="Blue")
f_1.pack(side=BOTTOM)
f_2.pack(side=BOTTOM)
button3 = Button(f_1, text=u"__В3 привіт", command=getF)
button3.pack(side=LEFT)

button4 = Button(f_2, text="_Б4 привіт", command=getF)
button4.pack(side=RIGHT) #LEFT)

canv = Canvas (root, width = 500, height = 500, bg = "white")

canv.create_line(500,1000,500,0,width=2,arrow=LAST)
canv.create_line(0,500,1000,500,width=2,arrow=LAST)

canv.create_line(200,50,300,50,width=3,fill="blue")
canv.create_line(0,0,100,100,width=2,arrow=LAST)

print('root.resizable()=<',root.resizable(),'>' )
print('root.root.maxsize()=<', root.maxsize(),'>' )

# кнопка за замовчуванням
button1 = Button(root, text=u"file read!", command=getF)
button1.pack(side=TOP) #LEFT) #'top') #place

# кнопка із зазначенням батьківського віджету та кількома
аргументами

button2 = Button(root, bg="red", text=u"Клікні мене!",
command=button_clicked)
button2.pack(side=TOP) #'top') #top')


root.protocol('WM_DELETE_WINDOW', window_deleted) # обробник
закриття вікна

#sys.exit(0)
```

```
First_x = -500;

forі in range(16000):
     if (i % 800 == 0):
          k = First_x + (1/16) * i
          canv.create_line(k + 500, -3 + 500, k + 500, 3 + 500,
width = 0.5, fill = 'black')
          canv.create_text(k + 515, -10 + 500, text = str(k),
                           fill="purple", font=("Helvectica", "10"))
          if (k! = 0):
               canv.create_line(-3 + 500, k + 500, 3 + 500,
                                k + 500, width = 0.5, fill = 'black')
               canv.create_text(20 + 500, k + 500, text = str(k),
                                fill="purple", font=("Helvectica", "10"))
     try:
          x = First_x + (1/16) * i
          new_f = f.replace('x', str(x))
          y = -eval(new_f) + 500
          x += 500
          # canv.create_oval(x, y, x + 1, y + 1, fill = 'black')
          canv.create_line(x, y, x + 1, y + 1, fill = 'black')
     except:
          pass
canv.pack()
################
root.bind('<Button-1>',b1)
root.bind('<Button-3>',b3)
root.bind('<Motion>',move)
root.bind('<Configure>',res)
#############
canv.focus_set()
root.mainloop()
print('@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@')
```

XXIX. Символьні обчислення мовою Python.

SymPy є відкритою бібліотекою символьних обчислень мовою Python. SymPy повністю написана мовою Python і не вимагає сторонніх бібліотек.

Нижче, у пізнавальній меті (за матеріалами сайту http://www.asmeurer.com/sympy_doc/dev-py3k/tutorial/tutorial.ru.html) , описані основні можливості пакету SymPy.

Детальніше дивитьсяhttp://www.sympy.org/en/index.htmlihttps://www.sympy.org/ru/index.html

Математична бібліотека Python SymPy

SymPy – це бібліотека Python для виконання символьних обчислень. Це система комп'ютерної алгебри, яка може виступати як окрема програма, так і як бібліотека для інших програм. Попрацювати з нею онлайн можна наhttps://live.sympy.org/. Бо це чистабібліотека Python, можна використовувати навіть в інтерактивному режимі.

У SymPy є різні функції, які застосовуються у сфері символьних обчислень, математичного аналізу, алгебри, дискретної математики, квантової фізики тощо. SymPy може представляти результат у різних форматах: LaTeX, MathML і таке інше. Поширюється бібліотека ліцензії New BSD. Першими цю бібліотеку випустили розробники Ondřej Čertík та Aaron Meurer у 2007 році.

Ось де застосовується SymPy:

- Багаточлени
- Математичний аналіз
- Дискретна математика
- Матриці
- Геометрія
- Побудова графіків
- Фізика
- Статистика
- Комбінаторика

Установка SymPy

Для роботи SymPy потрібна одна важлива бібліотека під назвою mpmath. Вона використовується для речової та комплексної арифметики з числами з плаваючою точкою довільної точності. Однак pip встановить її автоматично під час завантаження самої SymPy:

```
python -m pip install sympy
```

Використання SymPy як калькулятор

SymPy підтримує три типи чисельних даних: Float, Rational та Integer.

Rationalявляє собою звичайний дріб, який задається за допомогою двох цілих чисел: чисельник і знаменник. Наприклад, Rational(1, 2) є дріб 1/2, Rational(5, 2) є дріб 5/2, і так далі.

```
>>> від sympy import Rational
>>> a = Rational(1, 2)

>>> a
1/2
```

```
>>> a*2
1

>>> Rational(2)**50/Rational(10)**50
1/888178419700125232338905334472656 25
```

Важлива особливість Python-інтерпретатора - при розподілі двох "пітонівських" чисел типу int за допомогою оператора "/" виходить "пітонівський" тип float. Цей стандарт “true division” за замовчуванням включений і в sympy:

```
>>> 1/2
0.5
```

Зверніть увагу, що і в тому, і в іншому випадку ви маєте справу не з об'єктом Number з бібліотеки SymPy, який представляє число в SymPy, а з пітонівськими числами, які створюються самим інтерпретатором Python. Швидше за все, вам потрібно буде працювати з дробовими числами з бібліотеки SymPy, тому для того, щоб отримувати результат у вигляді об'єктів SymPy, переконайтеся, що ви використовуєте клас Rational. Комусь може здатися зручним позначати Rational як R:

```
import sympy
R = sympy.Rational
R(1, 2)
R(1)/2
```

У модулі Sympy є спеціальні константи, такі як e і pi, які поводяться як змінні (тобто вираз 1 + pi не перетворюється відразу в число, а так і залишиться 1 + pi):

```
>>> від sympy import pi, E
>>> pi**2
pi**2

>>> pi.evalf()
3.14159265358979

>>> (pi + E).evalf()
5.85987448204884
```

як видно, функція evalf переводить вихідний вираз у число з точкою, що плаває. Обчислення можна проводити з більшою точністю. Для цього потрібно передати як аргумент цієї функції необхідну кількість десяткових знаків:

```
>>> pi.evalf(100)
3.14159265358979323846264338327950288419716939937510582
09749445923078164062862089986280348253 82
>>>
```

Для роботи з математичною нескінченністю використовується символ oo:

```
>>> від sympy import oo
```

```
>>> oo > 99999
True
>>> oo + 1
oo
```

Змінні

На відміну від інших систем комп'ютерної алгебри, потрібно явно декларувати символьні змінні:

```
>>> від sympy import symbols
>>> x = symbols('x')
>>> y = symbols('y')
```

У лівій частині цього виразу знаходиться змінна Python, яка пітонівським присвоєнням співвідноситься з об'єктом класу Symbol із SymPy.

```
>>> з sympy.abc import x, theta
```

Символьні змінні можуть також задаватися і за допомогою функцій symbols або var. Вони допускають вказівку діапазону. Їхня відмінність полягає в тому, що var додає створені змінні в поточний простір імен:

```
>>> з символів import symbols, var
>>> a, b, c = symbols('a,b,c')
>>> d, e, f = symbols('d:f')
>>> var('g:h')
(g, h)
>>> var('g:2')
(g0, g1)
```

Примірники класу Symbol взаємодіють один з одним. Таким чином, за допомогою них конструюються алгебраїчні вирази:

```
>>> x + y + x - y
2*x

>>> (x + y)**2
(x + y)**2
#розкрити дужки
>>> ((x + y)**2).expand()
x**2 + 2*x*y + y**2
```

Змінні можуть бути замінені іншими змінними, числами або виразами за допомогою функції підстановки subs(old, new):

```
>>> ((x + y)**2).subs(x, 1)
(y + 1)**2

>>> ((x + y)**2).subs(x, y)
4*y**2

>>> ((x + y)**2).subs(x, 1 - y)
1
```

Тепер, з цього моменту, для всіх наведених нижче прикладів ми будемо припускати, що запустили наступну команду з налаштування системи відображення результатів (і використовуємо процедуру pprint):

```
>>> від sympy import init_printing
>>> init_printing(use_unicode=False, wrap_line=False,
no_global=True)
```

Вона надасть якіснішого відображення виразів. Докладніше про систему відображення та друку наведено нижче в розділі Друк. Якщо ж шрифт встановлений з юнікодом, то можна використовувати опцію use_unicode=True для ще красивішого виведення.

Алгебра

Щоб розкласти вираз на найпростіші дроби, використовується функція apart(expr, x):

```
#apart(expr, x):
from sympy import apart
from sympy.abc import x, y, z
from sympy import Integral, pprint
z= 1/( (x + 2)*(x + 1) )
pprint(z)
z=apart(1/( (x + 2)*(x + 1) ), x)
pprint(z)
z=apart((x + 1)/(x - 1), x)
pprint(z)
```

Висновок скрипту:

```
       1
---------------
(x + 1) * (x + 2)

    1 1
- ----- + -----
  x + 2 x + 1

      2
1 + -----
    x - 1

http://www.asmeurer.com/sympy_doc/dev-
py3k/tutorial/tutorial.ru.html
```

Щоб привести дріб до спільного знаменника, використовується функція together(expr, x):

```
>>> від sympy import together
>>> together(1/x + 1/y + 1/z)
x*y+x*z+y*z
---------------
     x*y*z

>>> together(apart((x + 1)/(x - 1), x), x)
x + 1
-----
x - 1

>>> together(apart(1/( (x + 2)*(x + 1) ), x), x)
       1
---------------
(x + 1) * (x + 2)
```

Обчислення

Межі

Для обчислення межі функції використовуйте функцію limit(function, variable, point).

```
Наприклад, щоб обчислити межу f(x) при x -> 0, потрібно
ввести limit(f, x, 0):
>>> від симпи import limit, Symbol, sin, oo
>>> x = Symbol("x")
>>> limit(sin(x)/x, x, 0)
1
```

також можна обчислювати межі при x, що прагне нескінченності:

```
>>> limit(x, x, oo)
oo

>>> limit(1/x, x, oo)
0

>>> limit(x**x, x, 0)
1
```

Більш складні приклади обчислення меж див.test_demidovich.py

Диференціювання

Продиференціювати будь-який вираз SymPy можна за допомогою diff(func, var). Приклади:

```
>>> від симпи import diff, Symbol, sin, tan
>>> x = Symbol('x')
>>> diff(sin(x), x)
cos(x)
>>> diff(sin(2*x), x)
```

```
2*cos(2*x)

>>> diff(tan(x), x)
   2
tan(x) + 1
```

Можна через межі перевірити правильність обчислень похідної:

```
>>> від sympy import limit
>>> від sympy.abc import delta
>>> limit((tan(x + delta) - tan(x))/delta, delta, 0)
   2
tan(x) + 1
```

Похідні вищих порядків можна обчислити, використовуючи додатковий параметр цієї функції diff(func, var, n):

```
>>> diff(sin(2*x), x, 1)
2*cos(2*x)

>>> diff(sin(2*x), x, 2)
-4*sin(2*x)

>>> diff(sin(2*x), x, 3)
-8*cos(2*x)
```

Розкладання в ряд

Для розкладання в ряд використовуйте метод series(var, point, order):

```
>>> від симпи імпорту Symbol, cos
>>> x = Symbol('x')
>>> cos(x).series(x, 0, 10)

     2 4 6 8
    xxxx/10\
1 - -- + -- - --- + ----- + O\x /
    2 24 720 40320

>>> (1/cos(x)).series(x, 0, 10)

     2 4 6 8
    x 5*x 61*x 277*x / 10\
1 + -- + ---- + ----- + ------ + O\x /
    2 24 720 8064
```

Ще один простий приклад:

```
>>> від симпи імпорту Integral, pprint

>>> y = Symbol("y")
>>> e = 1/(x + y)
```

```
>>> s = e.series(x, 0, 5)

>>> print(s)
1/y - x/y**2 + x**2/y**3 - x**3/y**4 + x**4/y**5 +
O(x**5)
>>> pprint(s)
          2 3 4
1 xxxx/5\
- - - + - - - + - + O\x /
y 2 3 4 5
    yyyy
```

Суми

Обчислює значення суми f(від заданої змінної) на заданих межах

summation(f, (i, a, b))- Знаходження суми доданків f, і змінюється від a до b:

$$\mathbf{summation(f, (i, a, b))} = \sum_{\mathbf{i = a}}^{\mathbf{b}} \mathbf{f}$$

Якщо він не може compute sum, його деталі відповідають summation formula. Repeated sums can computed by introducing additional limits:

Якщо сума не розраховується, то друкується відповідна формула:

```
>>> з sympy import summation, oo, symbols, log
>>> i, n, m = symbols('in m', integer=True)

>>> summation(2*i - 1, (i, 1, n))
2
n


>>> summation(1/2**i, (i, 0, oo))
2


>>> summation(1/log(n)**n, (n, 2, oo))
  oo
____
\   `
 \-n
 /log(n)
```

```
/__,
n = 2


>>> summation(i, (i, 0, n), (n, 0, m))
      3 2
mmm
-- + -- + -
6 2 3

>>> від sympy.abc import x
>>> з sympy import factorial
>>> summation(x**n/factorial(n), (n, 0, oo))
x
e
```

Інтегрування

SymPy підтримує обчислення певних та невизначених інтегралів за допомогою функції integrate(). Вона використовує розширений алгоритм Ріша-Нормана та деякі шаблони та евристики. Можна обчислювати інтеграли трансцендентних, простих та спеціальних функцій:

```
>>> від симпи імпорту integrate, erf, exp, sin, log,
oo, pi, sinh, symbols
>>> x, y = symbols('x,y')
```

Ви можете інтегрувати найпростіші функції:

```
>>> integrate(6*x**5, x)
6
x
>>> integrate(sin(x), x)
-cos(x)
>>> integrate(log(x), x)
x * log (x) - x
>>> integrate(2*x + sinh(x), x)
2
x + cosh(x)
```

Приклади інтегрування деяких спеціальних функцій:

```
>>> integrate(exp(-x**2)*erf(x), x)

  ____  2
\/ pi *erf (x)
--------------
      4
```

Можна також обчислити певний інтеграл:

```
>>> integrate(x**3, (x, -1, 1))
0
>>> integrate(sin(x), (x, 0, pi/2))
```

```
1
>>> integrate(cos(x), (x, -pi/2, pi/2))
2
```

Підтримуються і невласні інтеграли:

```
>>> integrate(exp(-x), (x, 0, oo))
1
>>> integrate(log(x), (x, 0, 1))
-1
```

Комплексні числа

Крім уявної одиниці I, що є уявним числом, символи теж можуть мати спеціальні атрибути (real, positive, complex тощо), які визначають поведінку цих символів при обчисленні символьних виразів:

```
>>> від симпи імпорту Symbol, exp, I
>>> x = Symbol("x") # a plain x with no attributes
>>> exp(I*x).expand()
I*x
e
>>> exp(I*x).expand(complex=True)

   -im(x) -im(x)
I * e * sin (re (x)) + e * cos (re (x))

>>> x = Symbol("x", real=True)
>>> exp(I*x).expand(complex=True)

I * sin (x) + cos (x)
```

Функції

тригонометричні

```
>>> від sympy import asin, asinh, cos, sin, sinh,
symbols, I
>>> x, y = symbols('x,y')

>>> sin(x + y).expand(trig=True)
sin(x)*cos(y) + sin(y)*cos(x)

>>> cos(x + y).expand(trig=True)
-sin(x)*sin(y) + cos(x)*cos(y)

>>> sin(I*x)
I*sinh(x)

>>> sinh(I*x)
I*sin(x)
```

```
>>> asinh(I)
I*pi
----
2

>>> asinh(I*x)
I*asin(x)

>>> sin(x).series(x, 0, 10)

     3 5 7 9
    xxxx/10\
x - -- + --- - ---- + ------ + Ox /
    6 120 5040 362880

>>> sinh(x).series(x, 0, 10)

     3 5 7 9
    xxxx/10\
x + -- + --- + ---- + ------ + Ox /
    6 120 5040 362880

>>> asin(x).series(x, 0, 10)

     3 5 7 9
    x 3*x 5*x 35*x / 10\
x + -- + ---- + ---- + ----- + Ox /
    6 40 112 1152

>>> asinh(x).series(x, 0, 10)

     3 5 7 9
    x 3*x 5*x 35*x / 10\
x - -- + ---- - ---- + ----- + Ox /
    6 40 112 1152
```

сферичні

```
>>> від sympy import Ylm
>>> з sympy.abc import theta, phi


>>> Ylm(1, 0, theta, phi)

  ___
```

```
\/ 3 *cos(theta)
----------------

    2*\/ pi

>>> Ylm(1, 1, theta, phi)

   ___ I*phi
-\/ 6 *e *sin(theta)
------------------------

        4*\/ pi

>>> Ylm(2, 1, theta, phi)

    ____ I*phi
-\/ 30 *e *sin(theta)*cos(theta)
-----------------------------------

              4*\/ pi
```

факторіали та гамма-функції

```
>>> від sympy import factorial, gamma, Symbol
>>> x = Symbol("x")
>>> n = Symbol("n", integer=True)

>>> factorial(x)
x!

>>> factorial(n)
n!

>>> gamma(x + 1).series(x, 0, 3) # ie factorial(x)

                      / 2  2  \
                    2 |EulerGamma  pi | / 3\
1 - EulerGamma*x + x *|------------ + ---| + Ox /
                      \  2     12/
```

діта-функції

```
>>> від sympy import zeta
>>> zeta(4, x)
zeta(4, x)

>>> zeta(4, 1)
```

```
  4
pi
---
90

>>> zeta(4, 2)

       4
     pi
-1 + ---
      90

>>> zeta(4, 3)

         4
  17 pi
- -- + ---
  16 90
```

багаточлени

```
>>> з імпорту import assoc_legendre, chebyshevt,
legendre, hermite
>>> chebyshevt(2, x)

   2
2*x - 1

>>> chebyshevt(4, x)

   4 2
8*x - 8*x + 1

>>> legendre(2, x)

   2
3*x 1
---- - -
2 2

>>> legendre(8, x)

      8 6 4 2
6435*x 3003*x 3465*x 315*x 35
------- - ------- + ------- - ------ + ---
  128 32 64 32 128
```

```
>>> assoc_legendre(2, 1, x)

        __________
       / 2
-3*x*\/ - x + 1


>>> assoc_legendre(2, 2, x)

     2
- 3*x + 3

>>> hermite(3, x)

   3
8*x - 12*x
```

Диференціальні рівняння

```
У isympy:
>>> від символів import Function, Symbol, dsolve
>>> f = Function('f')
>>> x = Symbol('x')
>>> f(x).diff(x, x) + f(x)
f(x) + Derivative(f(x), x, x)
>>> від симпи імпорту Integral, pprint
>>> uuu=f(x).diff(x, x) + f(x)
>>> pprint(uuu)

         2
        d
f(x) + ---(f(x))
         2
       dx

>>> dsolve(f(x).diff(x, x) + f(x), f(x))
Eq(f(x), C1*sin(x) + C2*cos(x))
>>> ud=dsolve(f(x).diff(x, x) + f(x), f(x))
>>> ud
Eq(f(x), C1*sin(x) + C2*cos(x))
>>> pprint(ud)
f(x) = C1*sin(x) + C2*cos(x)
```

Алгебраїчні рівняння

```
>>> від sympy import solve, symbols
```

```
>>> x, y = symbols('x,y')
>>> solve(x**4 - 1, x)
[-1, 1, -I, I]

>>> solve([x + 5*y - 2, -3*x + 6*y - 15], [x, y])
{x: -3, y: 1}
```

Лінійна алгебра

Матриці

Матриці задаються за допомогою конструктора Matrix:

```
>>> з імпорту import Matrix, Symbol
>>> Matrix([[1, 0], [0, 1]])
[1 0]
[]
[0 1]
```

У матрицях ви також можете використовувати символьні змінні:

```
>>> x = Symbol('x')
>>> y = Symbol('y')
>>> A = Matrix([[1, x], [y, 1]])
>>> A
[1 x]
[]
[y 1]

>>> A**2
[x * y + 1 2 * x]
[]
[ 2*yx*y + 1]
```

Для того, щоб дізнатися про матриці докладніше, прочитайте, будь ласка, Посібник з Лінійної Алгебри.

Зіставлення зі зразком

Щоб зіставити вирази зі зразками, використовуйте .match() разом із допоміжним класом Wild. Ця функція поверне словник із необхідними замінами, наприклад:

```
>>> від симпи імпорту Symbol, Wild
>>> x = Symbol('x')
>>> p = Wild('p')
>>> (5*x**2).match(p*x**2)
{p: 5}

>>> q = Wild('q')
>>> (x**2).match(p*x**q)
{p: 1, q: 2}
```

Якщо ж зіставлення не вдалося, функція поверне "None":

```
>>> print((x + 1).match(p**x))
None
```

Також можна використовувати параметр exclude для виключення деяких значень із результату:

```
>>> p = Wild('p', exclude=[1, x])
>>> print((x + 1).match(x + p)) # 1 is excluded
None
>>> print((x + 1).match(p + 1)) # x is excluded
None
>>> print((x + 1).match(x + 2 + p)) # -1 is not
excluded
{p_: -1}
```

Друк

Реалізовано кілька способів виведення виразів на екран.

Стандартний

Стандартний спосіб представлений функцією str(expression), яка працює наступним чином:

```
>>> from sympy import Integral
>>> від sympy.abc import x
>>> print(x**2)
x**2
>>> print(1/x)
1/x
>>> print(Integral(x**2, x))
Integral(x**2, x)
```

«Гарний друк»

Цей спосіб друку виразів заснований на ascii-графіці та реалізований через функцію pprint:

```
>>> від симпи імпорту Integral, pprint
>>> від sympy.abc import x
>>> pprint(x**2)

2
X

>>> pprint(1/x)

1
-
X
```

```
>>> pprint(Integral(x**2, x))
  /
|
| 2
| x dx
|
/
```

Якщо у вас встановлено шрифт із юнікодом, він буде використовувати Pretty-print із юнікодом за замовчуванням. Це налаштування можна вимкнути за допомогою use_unicode:

```
>>> pprint(Integral(x**2, x), use_unicode=True)
⌠
⌡ 2
  x dx
```

Для вивчення детальних прикладів роботи Pretty-print з юнікодом ви можете звернутися до статті Pretty Printing (https://github.com/sympy/sympy/wiki/Pretty-Printing) на Вікі.

Для того, щоб зробити pprint за замовчуванням у стандартному інтерпретаторі, виконуємо таку процедуру:

```
>>>від sympy import *
>>>import sys
>>>sys.displayhook = pprint
```

Приклади:

```
>>> від sympy import init_printing, var, Integral
>>> init_printing(use_unicode=False, wrap_line=False,
no_global=True)

>>> var("x")
X

>>> x**3/3

3
x
-
3

>>> Integral(x**2, x) #doctest: +NORMALIZE_WHITESPACE
  /
|
| 2
| x dx
```

```
|
/
```

Друк об'єктів Python

```
>>> від sympy.printing.python import python
>>> from sympy import Integral
>>> від sympy.abc import x
>>> print(python(x**2))
x = Symbol('x')
e = x**2
>>> print(python(1/x))
x = Symbol('x')
e = 1/x
>>> print(python(Integral(x**2, x)))
x = Symbol('x')
e = Integral(x**2, x)
```

Друк у форматі LaTeX

```
>>> від імпорту import Integral, latex
>>> від sympy.abc import x
>>> latex(x**2)
x^{2}
>>> latex(x**2, mode='inline')
$x^{2}$
>>> latex(x**2, mode='equation')
\begin{equation}x^{2}\end{equation}
>>> latex(x**2, mode='equation*')
\begin{equation*}x^{2}\end{equation*}
>>> latex(1/x)
\frac{1}{x}
>>> latex(Integral(x**2, x))
\int x^{2}\, dx
```

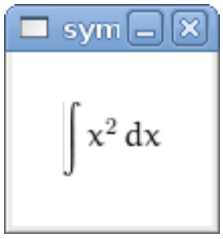
MathML

```
>>> від sympy.printing.mathml import mathml
>>> від імпорту import Integral, latex
>>> від sympy.abc import x
>>> print(mathml(x**2))
<apply><power/><ci>x</ci><cn>2</cn></apply>
>>> print(mathml(1/x))
<apply><power/><ci>x</ci><cn>-1</cn></apply>
```

Pyglet

```
>>> з імпорту import Integral, preview
>>> від sympy.abc import x
>>> preview(Integral(x**2, x))
```

З'явиться вікно pyglet з відмальованим виразом LaTeX:



Примітки
Якщо встановлені додаткові пакети.

isympyвикликає pprint автоматично, тому Pretty-print буде включено в isympy за замовчуванням.

Також доступний модуль друку – sympy.printing. Через цей модуль доступні наступні функції друку:
`pretty(expr), pretty_print(expr), pprint(expr) –`
Повертає або виводить на екран, відповідно, "Гарне" подання виразу expr.
`latex(expr), print_latex(expr) –`
Повертає або виводить на екран, відповідно,LaTeX-подання expr
`mathml(expr), print_mathml(expr) –`
Повертає або виводить на екран, відповідно, MathML (http://www.w3.org/Math/) -подання expr.
Щоб дізнатися про SymPy докладніше, зверніться:
`Посібник користувача SymPy`
`http://www.asmeurer.com/sympy_doc/dev-py3k/guide.html`

`та Опис модулів SymPy.http://www.asmeurer.com/sympy_doc/dev-py3k/modules/index.html`
Також можна звернутися на wiki.sympy.org - сайт, який містить безліч корисних прикладів, посібників та порад.

Приклади застосування пакету SymPy

SymPy - це пакет для символьних обчислень на пітоні, подібний до системи Mathematica. Він працює з виразами, що містять символи.

```
from sympy import *
init_printing()
```

Основними цеглинами, з яких будуються вирази, є символи. Символ має ім'я, яке використовується для друку виразів. Об'єкти класу Symbol потрібно створювати та присвоювати змінним пітонам, щоб їх можна було використовувати.

В принципі, ім'я символу та ім'я змінної, якою ми присвоюємо цей символ, - дві незалежні речі, і можна написати abc=Symbol('xyz').

Але тоді при введенні програми Ви будете використовувати abc, а під час друкування результатів SymPy буде використовувати xyz, що призведе до непотрібної плутанини. Тому краще, щоб ім'я символу збігалося з ім'ям змінної пітона, якому він надається.

У мовах, спеціально призначених для символьних обчислень, таких як Mathematica, якщо Ви використовуєте змінну, якої нічого не було присвоєно, вона автоматично сприймається як символ з тим же ім'ям. Пітон був спочатку призначений для символьних обчислень. Якщо Ви використовуєте змінну, якій нічого не було надано, Ви отримаєте повідомлення про помилку. Об'єкти типу Symbol слід створювати явно.

```
x=Symbol('x')
```

```
a=x**2-1
a
```

$$x^2 - 1$$

```
type(a)
```

```
sympy.core.add.Add
```

Можна визначити кілька символів одночасно. Рядок розбивається на імена за пробілами.

```
y,z=symbols('y z')
```

Підставимо замість x вираз y+1

```
a.subs(x,y+1)
```

$$(y+1)^2 - 1$$

Багаточлени та раціональні функції

SymPy не розкриває дужки автоматично. Для цього використовується функція expand.

```
a=(x+y-z)**6
a
```

$$(x+y-z)^6$$

```
a=expand(a)
a
```

$$x^6 + 6x^5y - 6x^5z + 15x^4y^2 - 30x^4yz + 15x^4z^2 + 20x^3y^3 - 60x^3y^2z + 60x^3yz^2 - 20x^3z^3 + 15x^2y^4 - 60x^2y^3z + 90x^2y^2z^2 - 60x^2yz^3 + 15x^2z^4 + 6xy^5 - 30xy^4z + 60xy^3z^2 - 60xy^2z^3 + 30xyz^4 - 6xz^5 + y^6 - 6y^5z + 15y^4z^2 - 20y^3z^3 + 15y^2z^4 - 6yz^5 + z^6$$

Ступінь многочлена a за x degree

```
degree(a,x)
```

6

Зберемо разом члени з певними ступенями x collect

```
collect(a,x)
```

$$x^6 + x^5(6y - 6z) + x^4(15y^2 - 30yz + 15z^2) + x^3(20y^3 - 60y^2z + 60yz^2 - 20z^3) + x^2(15y^4 - 60y^3z + 90y^2z^2 - 60yz^3 + 15z^4) + x(6y^5 - 30y^4z + 60y^3z^2 - 60y^2z^3 + 30yz^4 - 6z^5) + y^6 - 6y^5z + 15y^4z^2 - 20y^3z^3 + 15y^2z^4 - 6yz^5 + z^6$$

Багаточлен з цілими коефіцієнтами можна записати у вигляді добутку таких багаточленів (причому кожен співмножник вже неможливо розфакторизувати далі, залишаючись у рамках багаточленів з цілими коефіцієнтами).

Існують ефективні алгоритми для вирішення цієї задачі – factor.

```
a=factor(a)
a
```

$$(x + y - z)^6$$

SymPy не скорочує відносини багаточленів на їхній найбільший спільний дільник автоматично. Для цього використовується функція cancel.

```
a=(x**3-y**3)/(x**2-y**2)
a
```

$$\frac{x^3 - y^3}{x^2 - y^2}$$

```
cancel(a)
```

$$\frac{x^2 + xy + y^2}{x + y}$$

SymPy не призводить до суми раціональних виразів до спільного знаменника автоматично. Для цього використовується функція доіншого.

```
a=y/(x-y)+x/(x+y)
a
```

$$\frac{x}{x + y} + \frac{y}{x - y}$$

```
together(a)
```

$$\frac{x(x - y) + y(x + y)}{(x - y)(x + y)}$$

Функція simplify намагається переписати вираз у найпростішому вигляді. Це поняття немає чіткого визначення (у різних ситуаціях найпростішими можуть вважатися різні форми висловлювання), немає алгоритму такого спрощення.

Функція symplify працює евристично, і неможливо передбачити, які спрощення вона спробує зробити. Тому її зручно використовувати в інтерактивних сесіях, щоб подивитися, чи вдасться їй записати вираз у якомусь розумному вигляді, але небажано використовувати у програмах. Вони краще застосовувати більш спеціалізовані функції, які виконують одне певне перетворення висловлювання.

```
simplify(a)
```

$$\frac{x^2 + y^2}{x^2 - y^2}$$

Розкладання на елементарні дроби по відношенню до x та y - apart

```
apart(a,x)
```

$$-\frac{y}{x+y}+\frac{y}{x-y}+1$$

```
apart(a,y)
```

$$\frac{x}{x+y}+\frac{x}{x-y}-1$$

Підставимо конкретні чисельні значення замість змінних x та y subs

```
a=a.subs({x:1,y:2})
a
```

$$-\frac{5}{3}$$

А скільки це буде чисельно? Метод n()

```
a.n()
```

$$-1.66666666666667$$

Елементарні функції

SymPy автоматично застосовує спрощення елементарних функцій (які справедливі у всіх випадках).

```
sin(-x)
```

$$-\sin(x)$$

```
cos(pi/4),tan(5*pi/6)
```

$$\left(\frac{\sqrt{2}}{2},\quad -\frac{\sqrt{3}}{3}\right)$$

SymPy може працювати з числами з плаваючою точкою, що мають як завгодно велику точність. Ось π із 100 значущими цифрами – метод n(100).

```
pi.n(100)
```

$$3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534$$

E - це основа натуральних логарифмів.

```
log(1),log(E)
```

$$(0,\quad 1)$$

```
exp(log(x)),log(exp(x))
```

$$(x,\quad \log(e^x))$$

```
sqrt(0)
```

$$0$$

```
sqrt(x)**4,sqrt(x**4)
```

$$\left(x^2,\quad \sqrt{x^4}\right)$$

Символи можуть мати деякі властивості. Наприклад, вони можуть бути позитивними. Тоді SymPy може сильніше спростити квадратне коріння.

```
p,q=symbols('p q',positive=True)
sqrt(p**2)
```

$$p$$

```
sqrt(12*x**2*y),sqrt(12*p**2*y)
```

$$\left(2\sqrt{3}\sqrt{x^2 y},\quad 2\sqrt{3}p\sqrt{y}\right)$$

Нехай символ n буде цілим (I – це уявна одиниця).

```
n=Symbol('n',integer=True)
simplify(exp(2*pi*I*n))
```

$$1$$

```
sin(pi*n)
```

$$0$$

Метод rewrite намагається переписати вираз у термінах заданої функції.

```
cos(x).rewrite(exp),exp(I*x).rewrite(cos)
```

$$\left(\frac{e^{ix}}{2}+\frac{1}{2}e^{-ix},\quad i\sin(x)+\cos(x)\right)$$

```
asin(x).rewrite(log)
```

$$-i\log\left(ix+\sqrt{-x^2+1}\right)$$

Функція trigsimp намагається переписати тригонометричний вираз у найбільш простому вигляді. У програмах краще використовувати спеціалізовані функції.

```
trigsimp(2*sin(x)**2+3*cos(x)**2)
```

$$\cos^2(x)+2$$

Функція expand_trig розкладає синуси та косинуси сум та кратних кутів.

```
expand_trig(sin(x-y)),expand_trig(sin(2*x))
```

$$\left(\sin(x)\cos(y)-\sin(y)\cos(x),\quad 2\sin(x)\cos(x)\right)$$

Найчастіше потрібно зворотне перетворення - творів і ступенів синусів і косінусів у вирази, лінійні за цими функціями.

Наприклад, нехай ми працюємо з відрізком низки Фур'є.

```
a1,a2,b1,b2=symbols('a1 a2 b1 b2')
a=a1*cos(x)+a2*cos(2*x)+b1*sin(x)+b2*sin(2*x)
a
```

$$a_1\cos(x)+a_2\cos(2x)+b_1\sin(x)+b_2\sin(2x)$$

Ми хочемо звести його у квадрат і знову отримати відрізок ряду Фур'є.

```
a=(a**2).rewrite(exp).expand().rewrite(cos).expand()
a
```

$$\frac{a_1^2}{2}\cos(2x)+\frac{a_1^2}{2}+a_1a_2\cos(x)+a_1a_2\cos(3x)+a_1b_1\sin(2x)+a_1b_2\sin(x)+a_1b_2\sin(3x)+\frac{a_2^2}{2}\cos(4x)+\frac{a_2^2}{2}-a_2b_1\sin(x)+a_2b_1\sin(3x)+a_2b_2\sin(4x)-\frac{b_1^2}{2}\cos(2x)+\frac{b_1^2}{2}+b_1b_2\cos(x)-b_1b_2\cos(3x)-\frac{b_2^2}{2}\cos(4x)+\frac{b_2^2}{2}$$

```
a.collect([cos(x),cos(2*x),cos(3*x),sin(x),sin(2*x),sin(3*x)])
```

$$\frac{a_1^2}{2}+a_1b_1\sin(2x)+\frac{a_2^2}{2}\cos(4x)+\frac{a_2^2}{2}+a_2b_2\sin(4x)+\frac{b_1^2}{2}-\frac{b_2^2}{2}\cos(4x)+\frac{b_2^2}{2}+\left(\frac{a_1^2}{2}-\frac{b_1^2}{2}\right)\cos(2x)+(a_1a_2-b_1b_2)\cos(3x)+(a_1a_2+b_1b_2)\cos(x)+(a_1b_2-a_2b_1)\sin(x)+(a_1b_2+a_2b_1)\sin(3x)$$

Функція expand_log перетворює логарифми творів та ступенів у суми логарифмів (тільки для позитивних величин);

logcombineздійснює зворотне перетворення.

```
a=expand_log(log(p*q**2))
a
```

$$\log(p)+2\log(q)$$

```
logcombine(a)
```

$$\log(pq^2)$$

Функція expand_power_exp переписує ступеня, показники яких - суми через твори ступенів.

```
expand_power_exp(x**(p+q))
```

$$x^px^q$$

Функція expand_power_base переписує ступеня, основи яких – твори, через твори ступенів.

```
expand_power_base((x*y)**n)
```

$$x^ny^n$$

Функція powsimp виконує зворотні перетворення.

```
powsimp(exp(x)*exp(2*y)),powsimp(x**n*y**n)
```

$$\left(e^{x+2y},\quad (xy)^n\right)$$

Можна вводити функції користувача. Вони можуть мати довільну кількість аргументів.

```
f=Function('f')
f(x)+f(x,y)
```

$$f(x)+f(x,y)$$

Структура виразів

Внутрішнє уявлення виразу – це дерево. Функція srepr повертає рядок, що його представляє.

```
srepr(x+1)
```

```
"Add(Symbol('x'), Integer(1))"
```

```
srepr(x-1)
```

```
"Add(Symbol('x'), Integer(-1))"
```

```
srepr(x-y)
```

```
"Add(Symbol('x'), Mul(Integer(-1), Symbol('y')))"
```

```
srepr(2*x*y/3)
```

```
"Mul(Rational(2, 3), Symbol('x'), Symbol('y'))"
```

```
srepr(x/y)
```

```
"Mul(Symbol('x'), Pow(Symbol('y'), Integer(-1)))"
```

Замість бінарних операцій +, *, ** і т.д. можна використовувати функції Add, Mul, Pow тощо.

```
Mul(x,Pow(y,-1))==x/y
```

```
True
```

```
srepr(f(x,y))
```

```
"Function('f')(Symbol('x'), Symbol('y'))"
```

Атрибут func – це функція верхнього рівня у виразі, a args – список її аргументів.

```
a=2*x*y**2
a.func
```

```
sympy.core.mul.Mul
```

```
a.args
```

$$(2,\ x,\ y^2)$$

```
a.args[0]
```

$$2$$

```
for i in a.args:
    print(i)
```

```
2
x
y**2
```

Функція subs замінює змінну вираз.

```
a.subs(y,2)
```

$$8x$$

Вона може замінити кілька змінних. Для цього їй передається список кортежів чи словник.

```
a.subs([(x,pi),(y,2)])
```

8π

```
a.subs({x:pi,y:2})
```

8π

Вона може замінити не змінну, а подвираженіе - функцію з аргументами.

```
a=f(x)+f(y)
a.subs(f(y),1)
```

$f(x)+1$

```
(2*x*y*z).subs(x*y,z)
```

$2z^2$

```
(x+x**2+x**3+x**4).subs(x**2,y)
```

x^3+x+y^2+y

Підстановки виконуються послідовно. У разі спочатку x замінився на y, вийшло y3+y2; потім у цьому результаті y замінилося на x

```
a=x**2+y**3
a.subs([(x,y),(y,x)])
```

x^3+x^2

Якщо написати ці підстановки в іншому порядку, результат буде іншим.

```
a.subs([(y,x),(x,y)])
```

y^3+y^2

Але можна передати функції subs ключовий параметр simultaneous = True, тоді підстановки будуть виконуватися одночасно. Таким чином можна, наприклад, поміняти місцями x та y

```
a.subs([(x,y),(y,x)],simultaneous=True)
```

x^3+y^2

Можна замінити функцію іншою функцією.

```
g=Function('g')
a=f(x)+f(y)
a.subs(f,g)
```

$g(x)+g(y)$

Метод replace шукає подвыражения, відповідні зразку (який містить довільні змінні), і замінює їх у заданий вираз (воно може містити самі довільні змінні).

```
a=Wild('a')
(f(x)+f(x+y)).replace(f(a),a**2)
```

$$x^2 + (x+y)^2$$

```
(f(x,x)+f(x,y)).replace(f(a,a),a**2)
```

$$x^2 + f(x,y)$$

```
a=x**2+y**2
a.replace(x,x+1)
```

$$y^2 + (x+1)^2$$

Відповідати зразку має ціле подвыражение, це може бути частина співмножників у творі чи менша ступінь більшою.

```
a=2*x*y*z
a.replace(x*y,z)
```

$$2xyz$$

```
(x+x**2+x**3+x**4).replace(x**2,y)
```

$$x^4 + x^3 + x + y$$

Розв'язання рівнянь

```
a,b,c,d,e,f=symbols('a b c d e f')
```

Рівняння записується як функція Eq із двома параметрами. Функція solve повертає перелік рішень.

```
solve(Eq(a*x,b),x)
```

$$\left[\frac{b}{a}\right]$$

Втім, можна передати функції solve просто вираз. Мається на увазі рівняння, що цей вираз дорівнює 0.

```
solve(a*x+b,x)
```

$$\left[-\frac{b}{a}\right]$$

Квадратне рівняння має два рішення.

```
solve(a*x**2+b*x+c,x)
```

$$\left[\frac{1}{2a}\left(-b+\sqrt{-4ac+b^2}\right),\quad -\frac{1}{2a}\left(b+\sqrt{-4ac+b^2}\right)\right]$$

Система лінійних рівнянь.

```
solve([a*x+b*y-e,c*x+d*y-f],[x,y])
```

$$\left\{x:\frac{-bf+de}{ad-bc},\quad y:\frac{af-ce}{ad-bc}\right\}$$

Функція roots повертає коріння багаточлена зі своїми кратностями.

```
roots(x**3-3*x+2,x)
```

$$\{-2:1,\quad 1:2\}$$

Функція solve_poly_system вирішує систему поліноміальних рівнянь, будуючи їх базис Гребнера.

```
p1=x**2+y**2-1
p2=4*x*y-1
solve_poly_system([p1,p2],x,y)
```

$$\left[\left(4\left(-1-\sqrt{-\frac{\sqrt{3}}{4}+\frac{1}{2}}\right)\sqrt{-\frac{\sqrt{3}}{4}+\frac{1}{2}}\left(-\sqrt{-\frac{\sqrt{3}}{4}+\frac{1}{2}}+1\right),\quad -\sqrt{-\frac{\sqrt{3}}{4}+\frac{1}{2}}\right),\right.$$
$$\left(-4\left(-1+\sqrt{-\frac{\sqrt{3}}{4}+\frac{1}{2}}\right)\sqrt{-\frac{\sqrt{3}}{4}+\frac{1}{2}}\left(\sqrt{-\frac{\sqrt{3}}{4}+\frac{1}{2}}+1\right),\quad \sqrt{-\frac{\sqrt{3}}{4}+\frac{1}{2}}\right),$$
$$\left(4\left(-1-\sqrt{\frac{\sqrt{3}}{4}+\frac{1}{2}}\right)\sqrt{\frac{\sqrt{3}}{4}+\frac{1}{2}}\left(-\sqrt{\frac{\sqrt{3}}{4}+\frac{1}{2}}+1\right),\quad -\sqrt{\frac{\sqrt{3}}{4}+\frac{1}{2}}\right),$$
$$\left.\left(-4\left(-1+\sqrt{\frac{\sqrt{3}}{4}+\frac{1}{2}}\right)\sqrt{\frac{\sqrt{3}}{4}+\frac{1}{2}}\left(\sqrt{\frac{\sqrt{3}}{4}+\frac{1}{2}}+1\right),\quad \sqrt{\frac{\sqrt{3}}{4}+\frac{1}{2}}\right)\right]$$

Ряди

```
exp(x).series(x,0,5)
```

$$1+x+\frac{x^2}{2}+\frac{x^3}{6}+\frac{x^4}{24}+\mathcal{O}\left(x^5\right)$$

Ряд може починатися з негативного ступеня.

```
cot(x).series(x,n=5)
```

$$\frac{1}{x} - \frac{x}{3} - \frac{x^3}{45} + \mathcal{O}\left(x^5\right)$$

І навіть йти по підлозі цілими ступенями.

```
sqrt(x*(1-x)).series(x,n=5)
```

$$\sqrt{x} - \frac{x^{\frac{3}{2}}}{2} - \frac{x^{\frac{5}{2}}}{8} - \frac{x^{\frac{7}{2}}}{16} - \frac{5x^{\frac{9}{2}}}{128} + \mathcal{O}\left(x^5\right)$$

```
log(gamma(1+x)).series(x,n=6).rewrite(zeta)
```

$$-\gamma x + \frac{\pi^2 x^2}{12} - \frac{x^3\zeta\left(3\right)}{3} + \frac{\pi^4 x^4}{360} - \frac{x^5\zeta\left(5\right)}{5} + \mathcal{O}\left(x^6\right)$$

Підготуємо 3 ряди.

```
sinx=series(sin(x),x,0,8)
sinx
```

$$x - \frac{x^3}{6} + \frac{x^5}{120} - \frac{x^7}{5040} + \mathcal{O}\left(x^8\right)$$

```
cosx=series(cos(x),x,n=8)
cosx
```

$$1 - \frac{x^2}{2} + \frac{x^4}{24} - \frac{x^6}{720} + \mathcal{O}\left(x^8\right)$$

```
tanx=series(tan(x),x,n=8)
tanx
```

$$x + \frac{x^3}{3} + \frac{2x^5}{15} + \frac{17x^7}{315} + \mathcal{O}\left(x^8\right)$$

Твори та приватні ряди не обчислюються автоматично, до них треба застосувати функцію series.

```
series(tanx*cosx,n=8)
```

$$x - \frac{x^3}{6} + \frac{x^5}{120} - \frac{x^7}{5040} + \mathcal{O}\left(x^8\right)$$

```
series(sinx/cosx,n=8)
```

$$x + \frac{x^3}{3} + \frac{2x^5}{15} + \frac{17x^7}{315} + \mathcal{O}\left(x^8\right)$$

А цей ряд повинен дорівнювати 1. Але оскільки sinx і cosx відомі лише з обмеженою точністю, ми отримуємо 1 з тією ж точністю.

```
series(sinx**2+cosx**2,n=8)
```

$$1 + \mathcal{O}\left(x^8\right)$$

Тут перші члени скоротилися, і відповідь можна отримати лише з меншою точністю.

```
series((1-cosx)/x**2,n=6)
```

$$\frac{1}{2} - \frac{x^2}{24} + \frac{x^4}{720} + \mathcal{O}\left(x^6\right)$$

Ряди можна диференціювати та інтегрувати.

```
diff(sinx,x)
```

$$1 - \frac{x^2}{2} + \frac{x^4}{24} - \frac{x^6}{720} + \mathcal{O}\left(x^7\right)$$

```
integrate(cosx,x)
```

$$x - \frac{x^3}{6} + \frac{x^5}{120} - \frac{x^7}{5040} + \mathcal{O}\left(x^9\right)$$

Можна підставити ряд (якщо він починається з малого члена) замість змінного розкладання до іншого ряду. Ось ряди для sin(tan(x)) та tan(sin(x))

```
st=series(sinx.subs(x,tanx),n=8)
st
```

$$x + \frac{x^3}{6} - \frac{x^5}{40} - \frac{55x^7}{1008} + \mathcal{O}\left(x^8\right)$$

```
ts=series(tanx.subs(x,sinx),n=8)
ts
```

$$x + \frac{x^3}{6} - \frac{x^5}{40} - \frac{107x^7}{5040} + \mathcal{O}\left(x^8\right)$$

```
series(ts-st,n=8)
```

$$\frac{x^7}{30} + \mathcal{O}\left(x^8\right)$$

У ряд не можна підставляти чисельне значення змінної розкладання (отже, не можна будувати графік). Для цього потрібно спочатку прибрати $\mathcal{O}$ член, перетворивши відрізок низки багаточлен.

```
a=sinx.removeO()
```

```
a.subs(x,0.1)
```

0.0998334166468254

Похідні

```
a=x*sin(x+y)
diff(a,x)
```

$$x\cos(x+y)+\sin(x+y)$$

```
diff(a,y)
```

$$x\cos(x+y)$$

Друга похідна x і перша y

```
diff(a,x,2,y)
```

$$-x\cos(x+y)+2\sin(x+y)$$

Можна диференціювати вирази, що містять певні функції.

```
a=x*f(x**2)
b=diff(a,x)
b
```

$$2x^2\frac{d}{d\xi_1}f(\xi_1)\Big|_{\xi_1=x^2}+f(x^2)$$

Що це за «звір» такий вийшов?

```
print(b)
```

```
2*x**2*Subs(Derivative(f(_xi_1), _xi_1), (_xi_1,), (x**2,)) + f(x**2)
```

Функція Derivative є не обчисленою похідною. Її можна визначити методом doit.

```
a=Derivative(sin(x),x)
Eq(a,a.doit())
```

$$\frac{d}{dx}\sin(x)=\cos(x)$$

Інтеграли

```
integrate(1/(x*(x**2-2)**2),x)
```

$$\frac{1}{4}\log(x) - \frac{1}{8}\log(x^2-2) - \frac{1}{4x^2-8}$$

```
integrate(1/(exp(x)+1),x)
```

$$x - \log(e^x+1)$$

```
integrate(log(x),x)
```

$$x\log(x) - x$$

```
integrate(x*sin(x),x)
```

$$-x\cos(x) + \sin(x)$$

```
integrate(x*exp(-x**2),x)
```

$$-\frac{e^{-x^2}}{2}$$

```
a=integrate(x**x,x)
a
```

$$\int x^x\,dx$$

Вийшов невизначений інтеграл.

```
print(a)
```

```
Integral(x**x, x)
```

```
a=Integral(sin(x),x)
Eq(a,a.doit())
```

$$\int \sin(x)\,dx = -\cos(x)$$

Певні інтеграли.

```
integrate(sin(x),(x,0,pi))
```

$$2$$

oo - это ∞.

```
integrate(exp(-x**2),(x,0,oo))
```

$$\frac{\sqrt{\pi}}{2}$$

```
integrate(log(x)/(1-x),(x,0,1))
```

$$-\frac{\pi^2}{6}$$

Підсумовування рядів

```
summation(1/n**2,(n,1,oo))
```

$$\frac{\pi^2}{6}$$

```
summation((-1)**n/n**2,(n,1,oo))
```

$$-\frac{\pi^2}{12}$$

```
summation(1/n**4,(n,1,oo))
```

$$\frac{\pi^4}{90}$$

Чи не обчислена сума позначається Sum.

```
a=Sum(x**n/factorial(n),(n,0,oo))
Eq(a,a.doit())
```

$$\sum_{n=0}^{\infty} \frac{x^n}{n!} = e^x$$

Межі

```
limit((tan(sin(x))-sin(tan(x)))/x**7,x,0)
```

$$\frac{1}{30}$$

Це проста межа, він вважається розкладанням чисельника та знаменника в ряди. А якщо x=0, то з'являється особлива точка.

Знайдемо односторонні межі.

```
limit((tan(sin(x))-sin(tan(x)))/(x**7+exp(-1/x)),x,0,'+')
```

$$\frac{1}{30}$$

```
limit((tan(sin(x))-sin(tan(x)))/(x**7+exp(-1/x)),x,0,'-')
```

$$0$$

Диференціальні рівняння

```
t=Symbol('t')
x=Function('x')
p=Function('p')
```

Перший порядок.

```
dsolve(diff(x(t),t)+x(t),x(t))
```

$$x(t) = C_1 e^{-t}$$

Другий порядок.

```
dsolve(diff(x(t),t,2)+x(t),x(t))
```

$$x(t) = C_1 \sin(t) + C_2 \cos(t)$$

Система рівнянь першого ладу.

```
dsolve((diff(x(t),t)-p(t),diff(p(t),t)+x(t)))
```

$$[x(t) = C_1 \sin(t) + C_2 \cos(t), \quad p(t) = C_1 \cos(t) - C_2 \sin(t)]$$

Лінійна алгебра

```
a,b,c,d,e,f=symbols('a b c d e f')
```

Матрицю можна побудувати зі списку списків.

```
M=Matrix([[a,b,c],[d,e,f]])
M
```

$$\begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix}$$

```
M.shape
```

$$(2, \quad 3)$$

Матриця-рядок.

```
Matrix([[1,2,3]])
```

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$

Матриця-стовпець.

```
Matrix([1,2,3])
```

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Можна побудувати матрицю із функції.

```
def g(i,j):
    return Rational(1,i+j+1)
Matrix(3,3,g)
```

$$\begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{bmatrix}$$

Або з невизначеної функції.

```
g=Function('g')
M=Matrix(3,3,g)
M
```

$$\begin{bmatrix} g(0,0) & g(0,1) & g(0,2) \\ g(1,0) & g(1,1) & g(1,2) \\ g(2,0) & g(2,1) & g(2,2) \end{bmatrix}$$

```
M[1,2]
```

$$g(1,2)$$

```
M[1,2]=0
M
```

$$\begin{bmatrix} g(0,0) & g(0,1) & g(0,2) \\ g(1,0) & g(1,1) & 0 \\ g(2,0) & g(2,1) & g(2,2) \end{bmatrix}$$

```
M[2,:]
```

$$\begin{bmatrix} g(2,0) & g(2,1) & g(2,2) \end{bmatrix}$$

```
M[:,1]
```

$$\begin{bmatrix} g(0,1) \\ g(1,1) \\ g(2,1) \end{bmatrix}$$

```
M[0:2,1:3]
```

$$\begin{bmatrix} g(0,1) & g(0,2) \\ g(1,1) & 0 \end{bmatrix}$$

Поодинока матриця.

```
eye(3)
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Матриця з нулів.

```
zeros(3)
```

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

```
zeros(2,3)
```

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Діагональна матриця.

```
diag(1,2,3)
```

$$\begin{bmatrix}1 & 0 & 0\\0 & 2 & 0\\0 & 0 & 3\end{bmatrix}$$

```
M=Matrix([[a,1],[0,a]])
diag(1,M,2)
```

$$\begin{bmatrix}1 & 0 & 0 & 0\\0 & a & 1 & 0\\0 & 0 & a & 0\\0 & 0 & 0 & 2\end{bmatrix}$$

Операції із матрицями.

```
A=Matrix([[a,b],[c,d]])
B=Matrix([[1,2],[3,4]])
A+B
```

$$\begin{bmatrix}a+1 & b+2\\c+3 & d+4\end{bmatrix}$$

```
A*B,B*A
```

$$\left(\begin{bmatrix}a+3b & 2a+4b\\c+3d & 2c+4d\end{bmatrix}, \begin{bmatrix}a+2c & b+2d\\3a+4c & 3b+4d\end{bmatrix}\right)$$

```
A*B-B*A
```

$$\begin{bmatrix}3b-2c & 2a+3b-2d\\-3a-3c+3d & -3b+2c\end{bmatrix}$$

```
simplify(A**(-1))
```

$$\begin{bmatrix}\frac{d}{ad-bc} & -\frac{b}{ad-bc}\\-\frac{c}{ad-bc} & \frac{a}{ad-bc}\end{bmatrix}$$

```
det(A)
```

$$ad-bc$$

Власні значення та вектори

```
x=Symbol('x',real=True)
```

```
M=Matrix([[(1-x)**3*(3+x),4*x*(1-x**2),-2*(1-x**2)*(3-x)],
          [4*x*(1-x**2),-(1+x)**3*(3-x),2*(1-x**2)*(3+x)],
          [-2*(1-x**2)*(3-x),2*(1-x**2)*(3+x),16*x]])
M
```

$$\begin{bmatrix}(-x+1)^3(x+3) & 4x\left(-x^2+1\right) & (-x+3)\left(2x^2-2\right)\\4x\left(-x^2+1\right) & -(-x+3)(x+1)^3 & (x+3)\left(-2x^2+2\right)\\(-x+3)\left(2x^2-2\right) & (x+3)\left(-2x^2+2\right) & 16x\end{bmatrix}$$

```
det(M)
```

$$0$$

Значить, ця матриця має нульовий підпростір (вона звертає вектори з цього підпростору в 0). Базис цього підпростору.

```
v=M.nullspace()
len(v)
```

1

Воно одномірне.

```
v=simplify(v[0])
v
```

$$\begin{bmatrix} -\frac{2}{x-1} \\ \frac{2}{x+1} \\ 1 \end{bmatrix}$$

Проверим.

Перевіримо.

```
simplify(M*v)
```

$$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Власні значення та його кратності.

```
M.eigenvals()
```

$$\left\{0 : 1, \quad -\left(x^2+3\right)^2 : 1, \quad \left(x^2+3\right)^2 : 1\right\}$$

Якщо потрібні не лише власні значення, а й власні вектори, то потрібно використовувати метод eigenvects. Він повертає список кортежів. У кожному їх нульовий елемент - власне значення, перший - його кратність, і останній - перелік власних векторів, що утворюють базис (їх стільки, яка кратність).

```
v=M.eigenvects()
len(v)
```

3

```
for i in range(len(v)):
    v[i][2][0]=simplify(v[i][2][0])
v
```

$$\left[\left(0, \quad 1, \quad \left[\begin{bmatrix} -\frac{2}{x-1} \\ \frac{2}{x+1} \\ 1 \end{bmatrix}\right]\right), \quad \left(-\left(x^2+3\right)^2, \quad 1, \quad \left[\begin{bmatrix} \frac{x}{2}+\frac{1}{2} \\ \frac{x+1}{x-1} \\ 1 \end{bmatrix}\right]\right), \quad \left(\left(x^2+3\right)^2, \quad 1, \quad \left[\begin{bmatrix} \frac{x-1}{x+1} \\ -\frac{x}{2}+\frac{1}{2} \\ 1 \end{bmatrix}\right]\right)\right]$$

Перевіримо.

```
for i in range(len(v)):
    z=M*v[i][2][0]-v[i][0]*v[i][2][0]
    pprint(simplify(z))
```

$$\begin{bmatrix}0\\0\\0\\0\\0\\0\\0\end{bmatrix}$$

Жорданова нормальна форма

```
M=Matrix([[Rational(13,9),-Rational(2,9),Rational(1,3),Rational(4,9),Rational(2,3)],
          [-Rational(2,9),Rational(10,9),Rational(2,15),-Rational(2,9),-Rational(11,15)],
          [Rational(1,5),-Rational(2,5),Rational(41,25),-Rational(2,5),Rational(12,25)],
          [Rational(4,9),-Rational(2,9),Rational(14,15),Rational(13,9),-Rational(2,15)],
          [-Rational(4,15),Rational(8,15),Rational(12,25),Rational(8,15),Rational(34,25)]])
M
```

$$\begin{bmatrix}\frac{13}{9} & -\frac{2}{9} & \frac{1}{3} & \frac{4}{9} & \frac{2}{3}\\ -\frac{2}{9} & \frac{10}{9} & \frac{2}{15} & -\frac{2}{9} & -\frac{11}{15}\\ \frac{1}{5} & -\frac{2}{5} & \frac{41}{25} & -\frac{2}{5} & \frac{12}{25}\\ \frac{4}{9} & -\frac{2}{9} & \frac{14}{15} & \frac{13}{9} & -\frac{2}{15}\\ -\frac{4}{15} & \frac{8}{15} & \frac{12}{25} & \frac{8}{15} & \frac{34}{25}\end{bmatrix}$$

Метод M.jordan_form() повертає пару матриць, матрицю перетворення P та власне жорданову форму J: M=PJP−1

```
P,J=M.jordan_form()
J
```

$$\begin{bmatrix}1 & 0 & 0 & 0 & 0\\ 0 & 2 & 1 & 0 & 0\\ 0 & 0 & 2 & 0 & 0\\ 0 & 0 & 0 & 1-i & 0\\ 0 & 0 & 0 & 0 & 1+i\end{bmatrix}$$

```
P=simplify(P)
P
```

$$\begin{bmatrix} -2 & \frac{10}{9} & 0 & \frac{5i}{12} & -\frac{5i}{12} \\ -2 & -\frac{5}{9} & 0 & -\frac{5i}{6} & \frac{5i}{6} \\ 0 & 0 & \frac{4}{3} & -\frac{3}{4} & -\frac{3}{4} \\ 1 & \frac{10}{9} & 0 & -\frac{5i}{6} & \frac{5i}{6} \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

Перевіримо.

```
Z=P*J*P**(-1)-M
simplify(Z)
```

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

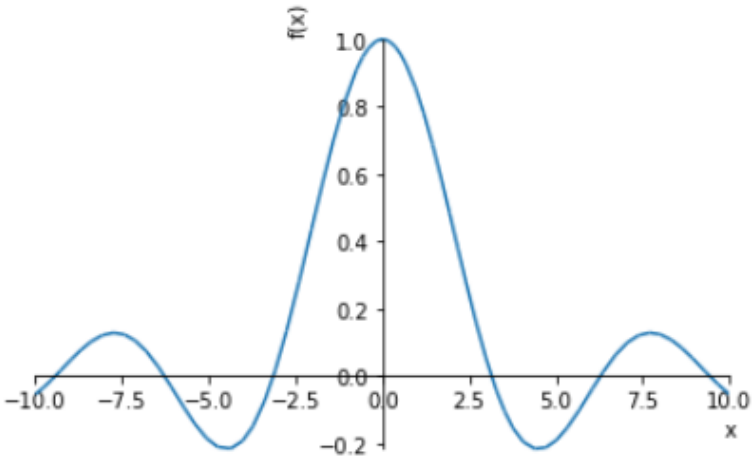
Графіки

SymPy використовує matplotlib. Однак він розподіляє точки по x адаптивно, а не рівномірно.

%matplotlib inline

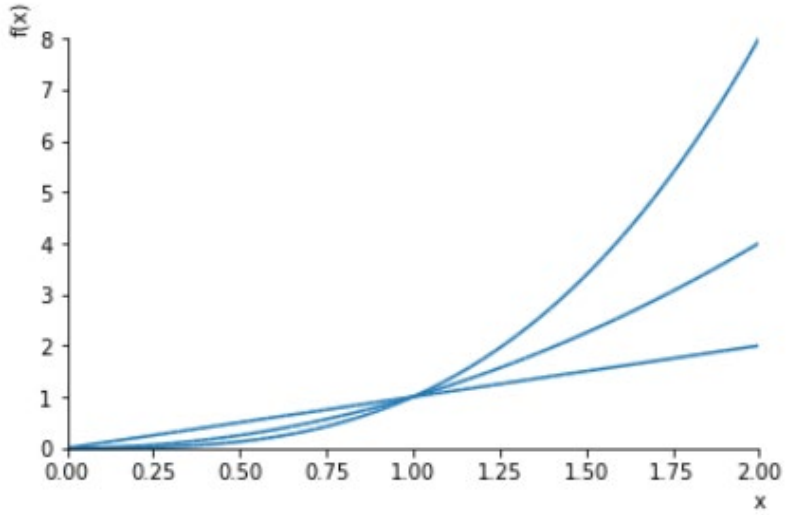
Одна функція.

plot(sin(x)/x,(x,-10,10))



Декілька функцій.

plot(x,x**2,x**3,(x,0,2))

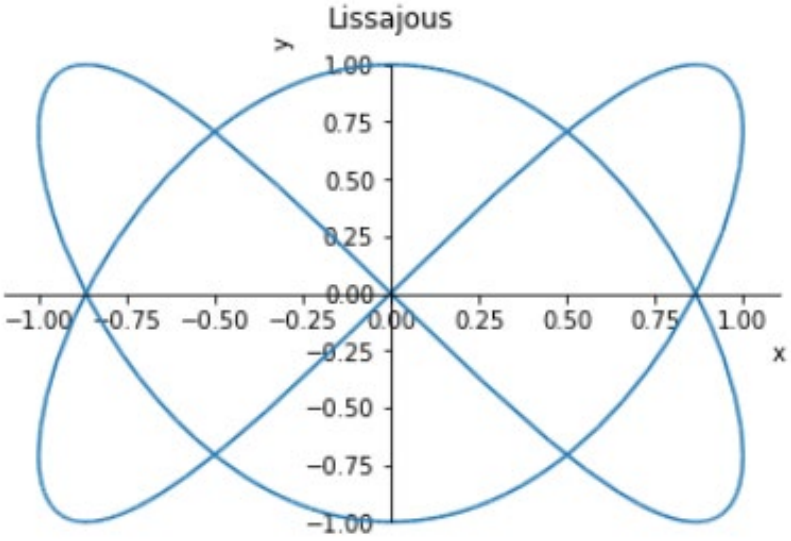


Інші функції слід імпортувати з пакету sympy.plotting.

```
від sympy.plotting import
(plot_parametric,plot_implicit,
plot3d,plot3d_parametric_line,
plot3d_parametric_surface)
```

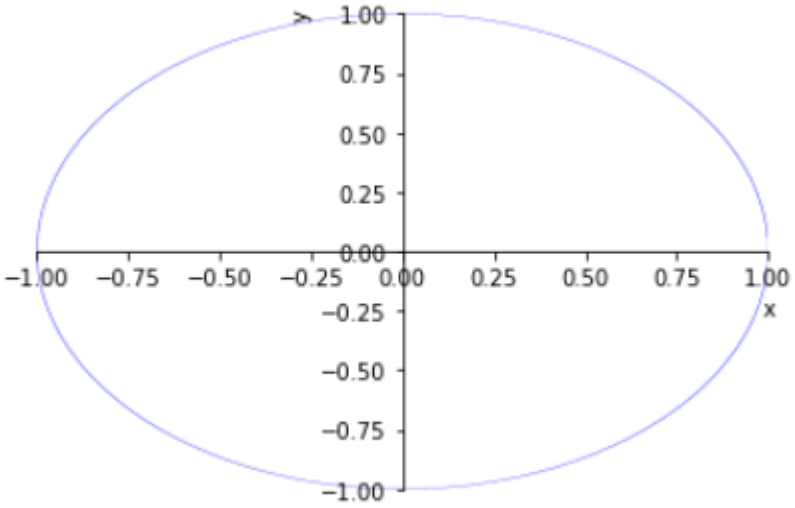
Параметричний графік- фігура Ліссажу.

```
t=Symbol('t')
plot_parametric(sin(2*t),cos(3*t),(t,0,2*pi),
          title='Lissajous',xlabel='x',ylabel='y')
```



Неявний графік- Коло.

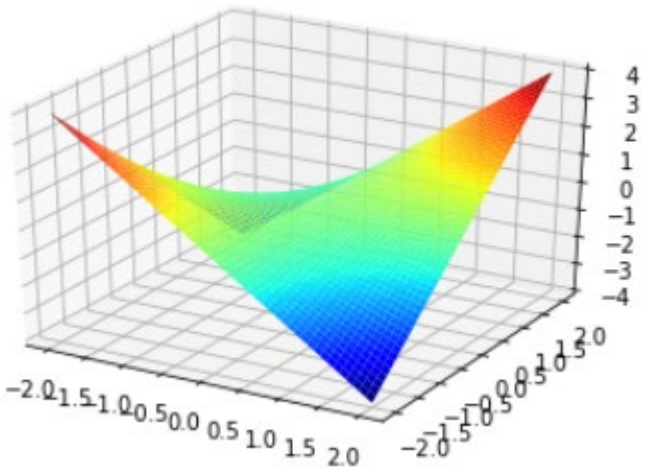
```
plot_implicit(x**2+y**2-1,(x,-1,1),(y,-1,1))
```



Поверхня.

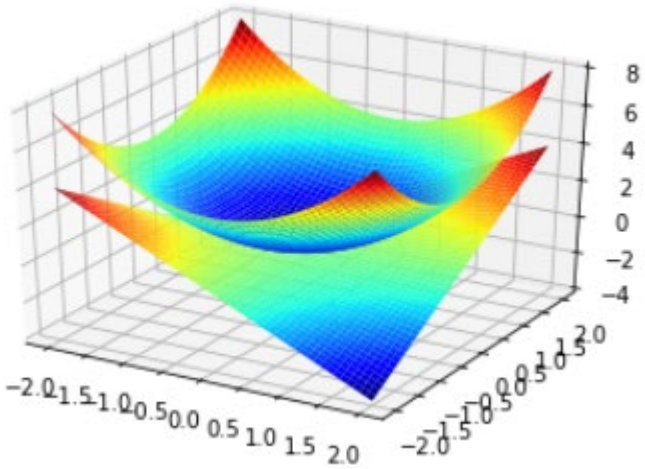
Якщо вона будується не inline, а в окремому вікні, її можна крутити мишкою.

```
plot3d(x*y,(x,-2,2),(y,-2,2))
```



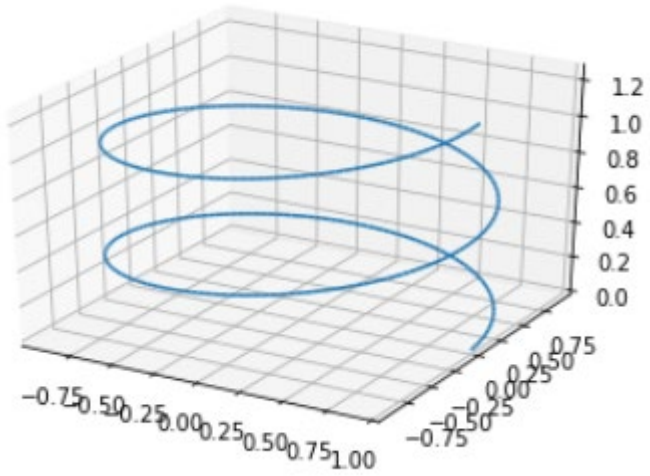
Декілька поверхонь.

plot3d(x**2+y**2,x*y,(x,-2,2),(y,-2,2))



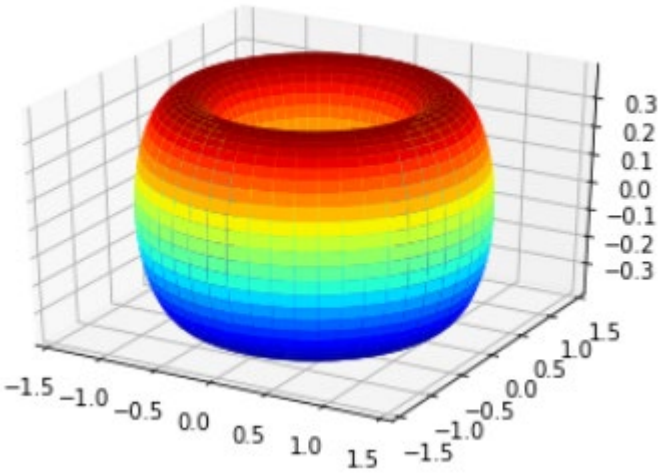
Параметрична просторова лінія- Спіраль.

```
a=0.1
plot3d_parametric_line(cos(t),sin(t),a*t,(t,0,4*pi))
```



Параметрична поверхня – тор.

```
u,v=symbols('u v')
a=0.4
plot3d_parametric_surface((1+a*cos(u))*cos(v),
                          (1+a*cos(u))*sin(v),a*sin(u),
                          (u, 0,2 * pi), (v, 0,2 * pi))
```



(За https://www.inp.nsk.su/~grozin/python/b25_sympy.html)

XXX. Список використаних джерел

Нижче перераховані адреси сайтів, які стали основою для написання цього посібника.

1. https://pythonworld.ru/numpy
2. https://docs.scipy.org/doc/numpy/reference/generated/numpy.set_printoptions.html

3. https://docs.scipy.org/doc/numpy/reference/
4. https://docs.scipy.org/doc/numpy/reference/routines.math.html
5. https://pythonworld.ru/samouchitel-python
6. http://old.pynsk.ru/posts/2015/Nov/09/matematika-v-python-preobrazovanie-fure/
7. https://github.com/whitehorn/Scientific_graphics_in_python
8. https://metanit.com/python/
9. http://nbviewer.jupyter.org/github/whitehorn/Scientific_graphics_in_python/tree/master/
10. http://jenyay.net/Programming/Python
11. https://habrahabr.ru/company/mailru/blog/228379/)/
12. https://github.com/sympy/sympy/wiki/Pretty-Printing
13. http://www.asmeurer.com/sympy_doc/dev-py3k/tutorial/tutorial.ru.html
14. http://www.realcoding.net/articles
15. http://docplayer.ru/51061131-Vvedení-v-nauchnyy-python.html
16. https://ua.wikiversity.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC %D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%B8_%D0%BD%D0%B0%
D1%83%D1%87%D0%BD%D1%8B%D0%B5_%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%BD%D0%B0_%D1%8F%D0%B7%D1%8B%D0%BA%D0%B5_Python/%C2%A75
17. https://eax.me/python-matplotlib
18. http://jenyay.net/Matplotlib/Widgets
19. https://matplotlib.org/2.0.0/examples/animation/animate_decay.html
20. http://www.asmeurer.com/sympy_doc/dev-py3k/tutorial/tutorial.ru.html
21. http://www.sympy.org/en/index.html