Тема 9. Модулі sys, itertools, locale, datetime та модуль logging

I. 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).

Прикладник вивести 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	'os2emx'

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.

II. 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-
```

III. 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:

```
import locallocale.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')
```

IV. 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:

```
import datetime

yesterday = datetime.date(2017,5, 2)
```

```
print(yesterday) # 2017-05-02
```

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

```
from datetime 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.

```
from datetime 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:

```
from datetime 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))
```

V. 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.