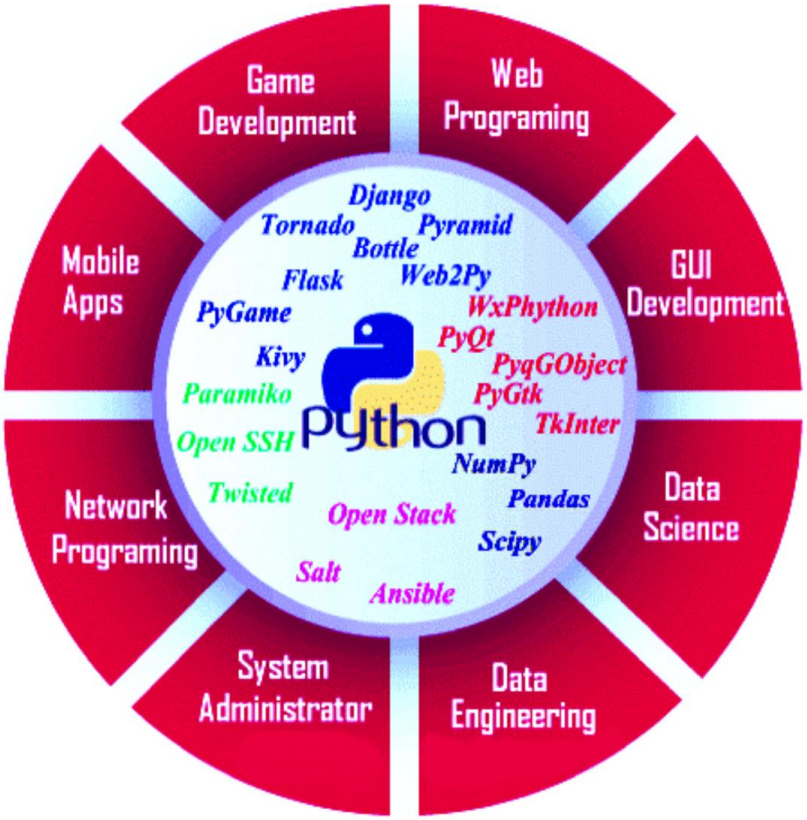
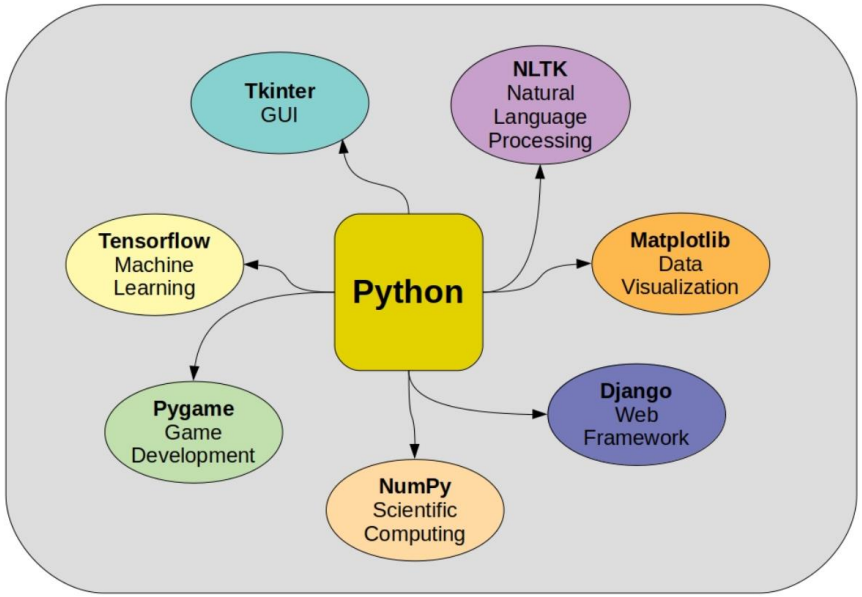
Програмування на мові Python

- Лекцій – 28 годин
- Лабораторних робіт – 14 годин





Мова програмування Python

- Python – інтерпретована, мультіпарадігменна мова програмування високого рівня з динамічною типізацією, автоматичним управлінням пам'яттю і зручними високорівневими структурами даних.
- Розроблена в 1991 році Гвідо ван Россумом.
- Підтримує **обробку винятків**, **паралельні обчислення**.
- Підтримується **декілька парадигм програмування**, зокрема: об'єктно-орієнтована, процедурна, функціональна та аспектно-орієнтована.

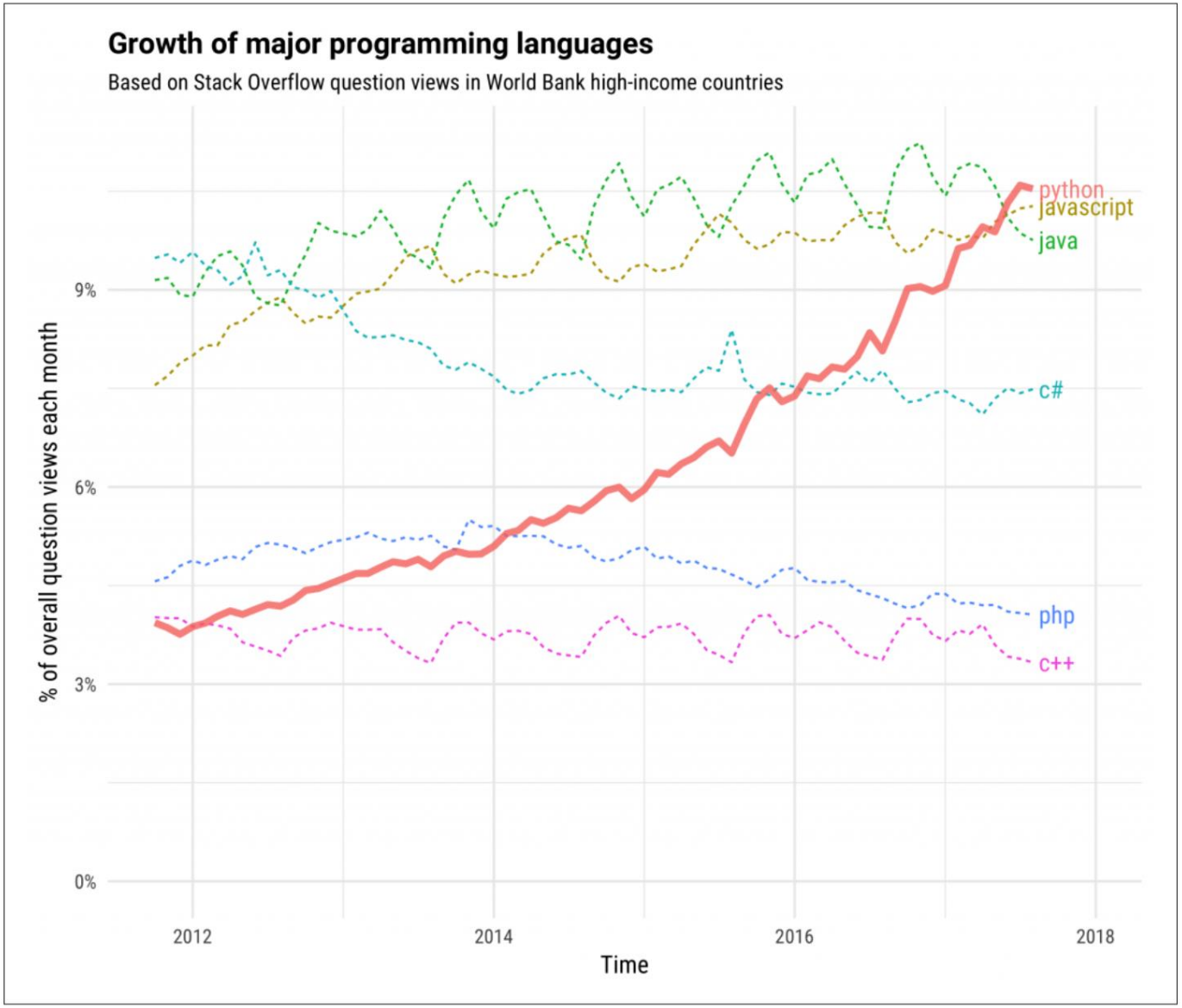
Парадигми програмування

- **Імперативна** - описує процес обчислення у вигляді інструкцій, що змінюють стан програми
- **Об'єктно-орієнтована** - основною концепцією є поняття об'єкта, що ототожнюється з об'єктом предметної області
- **Функціональна** - в ній процес обчислення трактується як обчислення значень функцій в математичному розумінні (спосіб вирішення задачі описується у вигляді залежності функцій одна від одної)

Why Python?

- **Найшвидше зростаюча** основна мова програмування, як ви можете бачити на малюнку Figure 1-4.
- Редакція Індексу TIOBE (https://www.tiobe.com/tiobe-index/) у червні 2019 року каже: «Цього місяця Python знову досяг рекордно високого показника TIOBE - 8,5%. Якщо Python зможе утримати цей темп, він, ймовірно, замінить C та Java через 3–4 роки, ставши, таким чином, **найпопулярнішою мовою програмування у світі** ».
- Мова програмування 2018 року (TIOBE), а також **найкращий рейтинг** IEEE Spectrum та PyPL.
- . . . . . .

Figure 1-4. Python leads in major programming language growth (Bill Lubanovic Introducing Python. – 2020)



. Попівщий

TIOBE Index for September 2015

Sep 2015	Sep 2014	Change	Programming Language	Ratings	Change
1	2	↑	Java	19.565%	+5.43%
2	1	↓	C	15.621%	-1.10%
3	4	↑	C++	6.782%	+2.11%
4	5	↑	C#	4.909%	+0.56%
5	8	↑	Python	3.664%	+0.88%
6	7	↑	PHP	2.530%	-0.59%
7	9	↑	JavaScript	2.342%	-0.11%
8	11	↑	Visual Basic .NET	2.062%	+0.53%
9	12	↑	Perl	1.899%	+0.53%
10	3	⇊	Objective-C	1.821%	-8.11%
11	29	⇈	Assembly language	1.806%	+1.22%
12	13	↑	Ruby	1.783%	+0.50%
13	15	↑	Delphi/Object Pascal	1.745%	+0.59%
14	14		Visual Basic	1.532%	+0.26%
15	17	↑	Pascal	1.298%	+0.40%
16	18	↑	Swift	1.188%	+0.34%

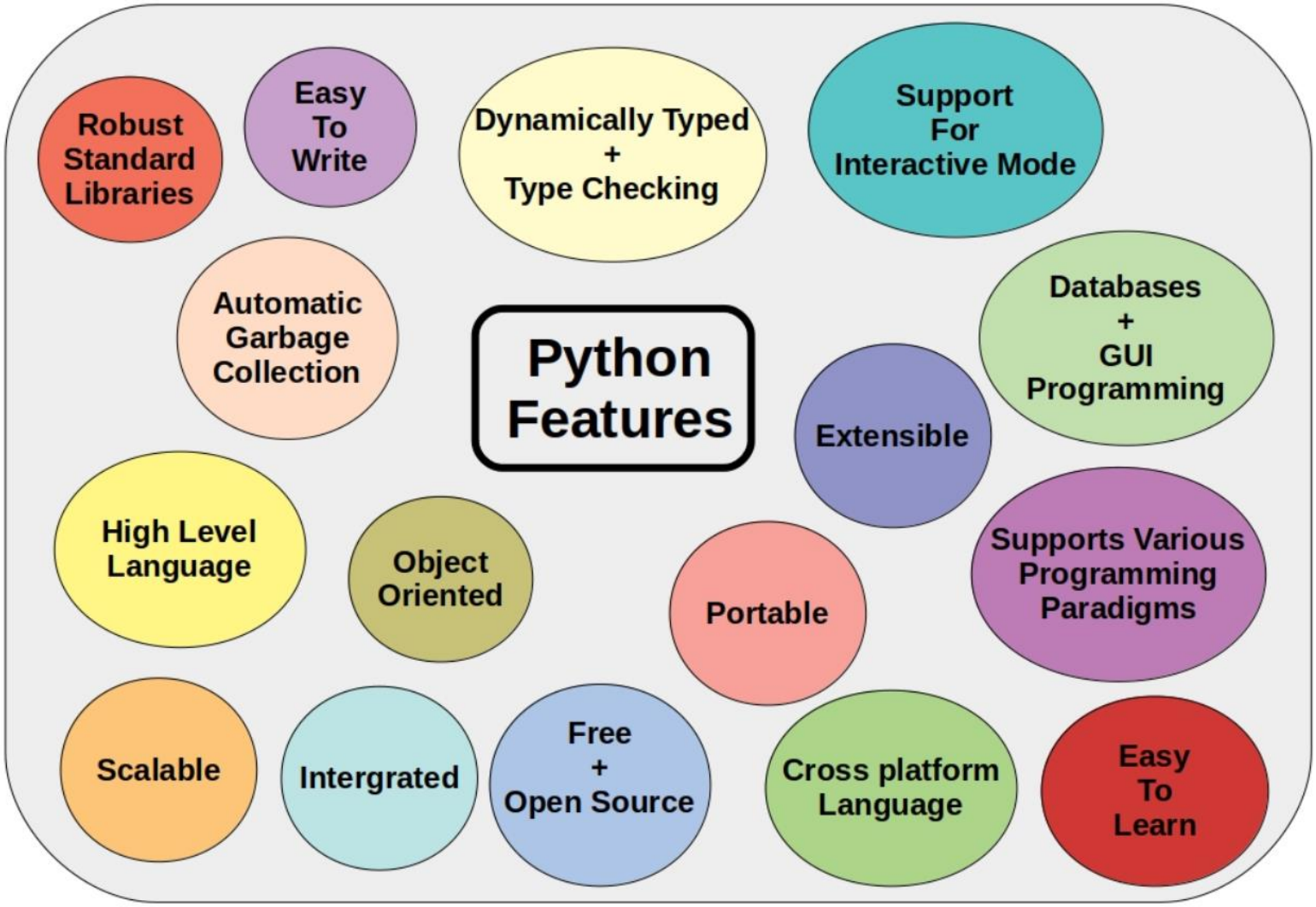
TIOBE Index for April 2021

Apr 2021	Apr 2020	Change	Programming Language	Ratings
1	2	↑	C	14.32%
2	1	↓	Java	11.23%
3	3		Python	11.03%
4	4		C++	7.14%
5	5		C#	4.91%
6	6		Visual Basic	4.55%
7	**7**		**JavaScript**	**2.44%**
8	14	⇈	Assembly language	2.32%
9	8	↓	PHP	1.84%
10	9	↓	SQL	1.83%
11	19	⇈	Classic Visual Basic	1.54%
12	22	⇈	Delphi/Object Pascal	1.47%
13	13		Ruby	1.23%
14	12	↓	Go	1.22%
15	11	⇊	Swift	1.19%
16	10	⇊	R	1.12%

TIOBE Index for August 2021

Aug 2021	Aug 2020	Change	Programming Language	Ratings
1	1		C	12.57%
2	3	▲	Python	11.86%
3	2	▼	Java	10.43%
4	4		C++	7.36%
5	5		C#	5.14%

https://www.tiobe.com/tiobe-index/


Python Features
Robust Standard Libraries
Easy To Write
Dynamically Typed + Type Checking
Support For Interactive Mode
Automatic Garbage Collection
Databases + GUI Programming
Extensible
High Level Language
Object Oriented
Portable
Supports Various Programming Paradigms
Scalable
Intergrated
Free + Open Source
Cross platform Language
Easy To Learn

Why Not Python?

- Python - не найкраща мова для будь-якої ситуації.
- Вона досить швидка для більшості програм, але може бути недостатньо швидкою для деяких із більш вимогливих.
- Якщо ваша програма більшу частину свого часу проводить за вирішенням обчислювальних задач (пов'язаних з навантаженням процесора), програма, написана на C, C ++, C#, Java, Rust або Go, як правило, працюватиме швидше, ніж її еквівалент на Python.
- . . . . . .

(Дивись Bill Lubanovic Introducing Python. – 2020 p.16)

Динамічна типізація

- **Немає попереднього оголошення типів** – тип змінної виводиться в процесі виконання

```
# Функція може повернути об'єкт будь-якого типу
result = f (x)
```

- **строга типізація**

  Неприпустимо: 5 + "3"

Гвідо ван Россум

- **Guido van Rossum** - голландський програміст
- Головний автор навчальної мови ABC (90-ті)
- Серед програмістів: «Великодушний довічний диктатор» (англ. Benevolent Dictator For Life, скор. BDFL)

Benevolent dictator for life

Benevolent dictator for life (**BDFL**) is a title given to a small number of open-source software development leaders, typically project founders who retain the final say in disputes or arguments within the community.

Persons sometimes referred to as "benevolent dictators for life"

Name	Project	Type
Dries Buytaert	Drupal	content management framework
Rasmus Lerdorf	PHP	scripting language
Guido van Rossum	Python	programming language
Linus Torvalds	Linux	operating system kernel
Patrick Volkerding	Slackware	Linux distribution
Larry Wall	Perl	programming language

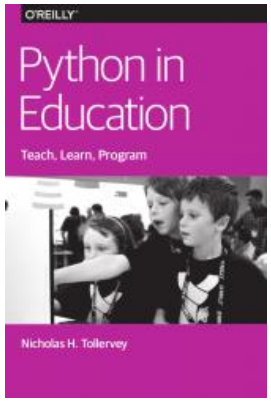
Історія Python

- З'явився у 1991 році
- Названий на честь британського серіалу 70-х "Літаючий цирк Монті Пайтона"
- Python 1.0 – January 1994
- Python 2.0 – October 16, 2000
- Python 3.0 – December 3, 2008
- Поточні версії (вересень 2021) :
  - Python 2.7.17
  - Python 3.9.7

Вплив інших мов

- **ABC** - відступи в угрупованні операторів, високорівневі структури даних
- **Modula-3** - пакети, модулі, іменовані аргументи функцій
- **C, C ++** - основні конструкції
- **Smalltalk** - об'єктно-орієнтоване програмування
- **Lisp** - деякі риси функціонального програмування
- **Fortran** - зрізи масивів, комплексна арифметика
- **Java** - шаблони, винятки, ...

Python в освіті

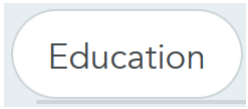


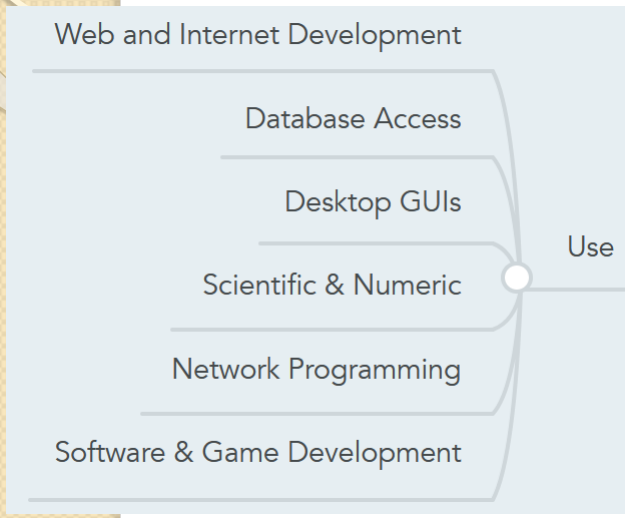
- Найбільше розповсюдження Python отримав у США
- Це перша мова програмування в MIT (*Massachusetts Institute of Technology*) – флагмані навчання *computer science* в світі
- Python рекомендований як мова програмування для шкіл та перша мова програмування в університетах США

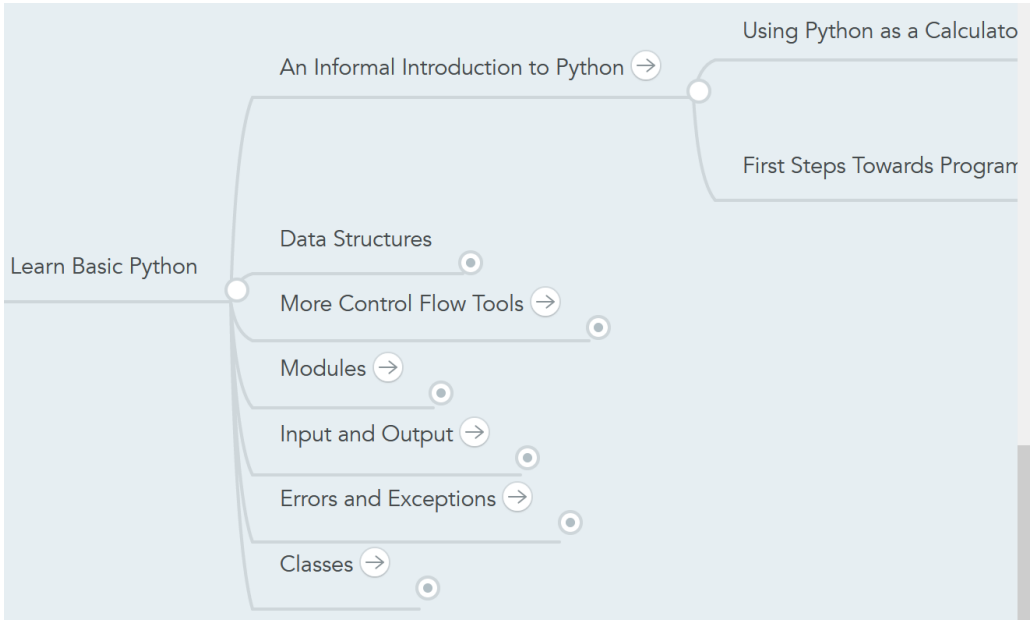
Бібліотеки Python

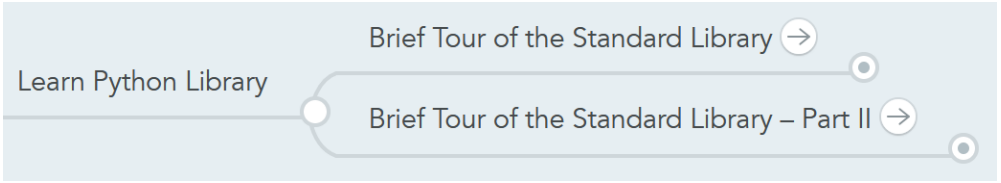
- **Потужна вбудована бібліотека**
  - Робота з Web, регулярні вирази, архіви, багатозадачність, UI
- **Велика кількість Python-інтерфейсів для популярних бібліотек**
  - 2D і 3D графіка, OpenGL, DirectX
  - робота з базами даних, MySQL, PostgreSQL
  - робота з мультимедіа: звук, відео, зображення
  - розробка користувальницьких інтерфейсів, Qt, Gtk, WxWidgets

Python Mind Map

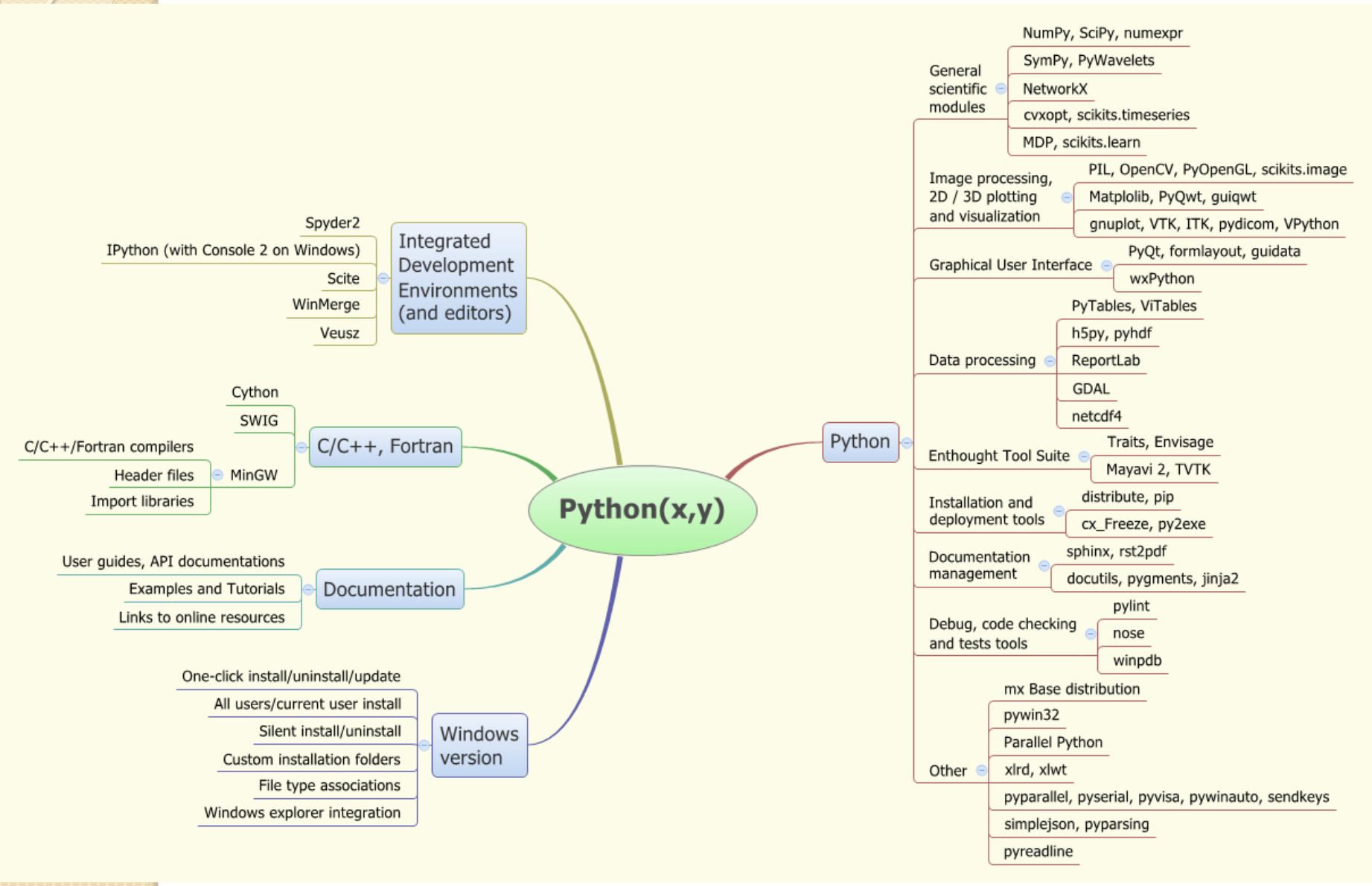








- https://www.mindmeister.com/752422209/python?fullscreen=1


Python(x,y)
Integrated Development Environments (and editors)
Spyder2
IPython (with Console 2 on Windows)
Scite
WinMerge
Veusz
C/C++, Fortran
Cython
SWIG
MinGW
C/C++/Fortran compilers
Header files
Import libraries
Documentation
User guides, API documentations
Examples and Tutorials
Links to online resources
Windows version
One-click install/uninstall/update
All users/current user install
Silent install/uninstall
Custom installation folders
File type associations
Windows explorer integration
Python
General scientific modules
NumPy, SciPy, numexpr
SymPy, PyWavelets
NetworkX
cvxopt, scikits.timeseries
MDP, scikits.learn
Image processing, 2D / 3D plotting and visualization
PIL, OpenCV, PyOpenGL, scikits.image
Matplolib, PyQwt, guiqwt
gnuplot, VTK, ITK, pydicom, VPython
Graphical User Interface
PyQt, formlayout, guidata
wxPython
Data processing
PyTables, ViTables
h5py, pyhdf
ReportLab
GDAL
netcdf4
Enthought Tool Suite
Traits, Envisage
Mayavi 2, TVTK
Installation and deployment tools
distribute, pip
cx_Freeze, py2exe
Documentation management
sphinx, rst2pdf
docutils, pygments, jinja2
Debug, code checking and tests tools
pylint
nose
winpdb
Other
mx Base distribution
pywin32
Parallel Python
xlrd, xlwt
pyparallel, pyserial, pyvisa, pywinauto, sendkeys
simplejson, pyparsing
pyreadline

Застосування Python

- **Інтерактивна консоль** - потужний «калькулятор»
  - робота з числами, матрицями, файлами, зображеннями, статистичного аналізу та ін.
- **Мова програмування для невеликих скриптів**
  - обробка зображень, створення резервних копій
- **Мова програмування для прототипування**
  - швидке створення шаблону програми з UI
  - швидка перевірка роботи алгоритму
- **Мова програмування для повноцінних програм**
  - Gajim, BitTorrent, Dropbox, EVE Online

Застосування Python

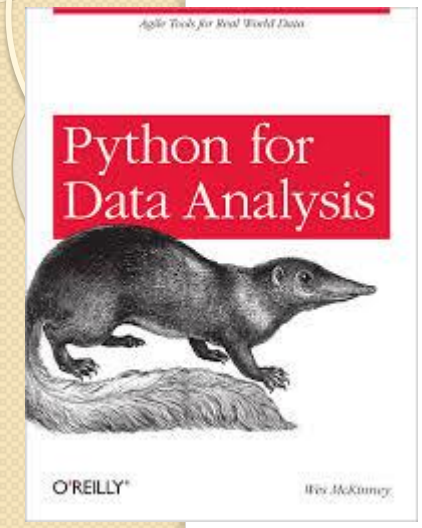
- **Мова програмування для web-застосунків**
  - багато фреймворків (**Django**, Flask, TurboGears, web2py, Pylons, Zope, WebWare), підтримують швидке створення повнофункціональних і якісних сайтів на основі Python
- **Вбудована в додатки мова програмування**
  - **Вбудована Python-консоль в якій можна оперувати з об'єктами додатків мовою Python**
    - **3D моделювання**: Blender, Maya
    - **Обробка зображень**: GIMP
    - **Робота з ГІС даними**: ESRI ArcGIS
    - **Математичні пакети**: Sage, IPython Notebook
    - **Офісний пакет** OpenOffice

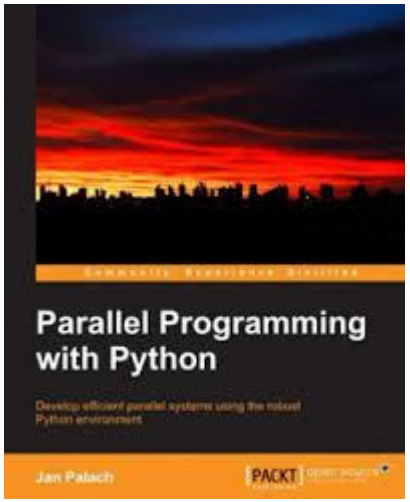
Приклади використання Python

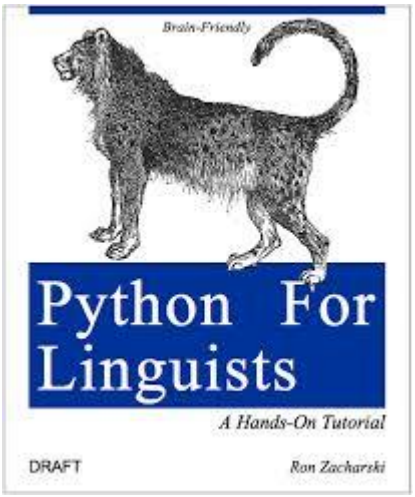
- Компанія **Google** використовує Python в своїй **пошуковій системі** і оплачує працю творця Python - Гвідо ван Россума
- Такі компанії, як **Intel, Cisco, Hewlett-Packard, Seagate, Qualcomm і IBM**, використовують Python для **тестування апаратного забезпечення**
- Служба колективного використання **відеоматеріалів YouTube** в значній мірі реалізована на Python
- **NSA** використовує Python для **шифрування і аналізу розвідданих**
- The **Raspberry Pi** single-board computer promotes Python as its educational language.

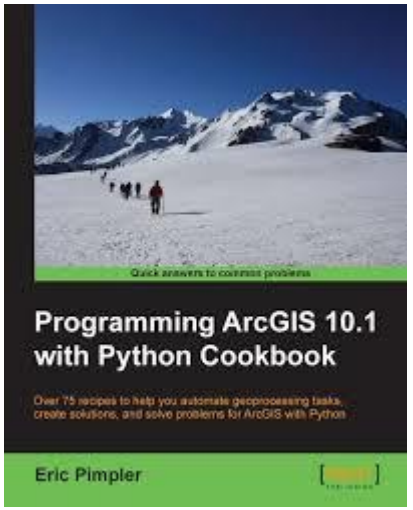
Приклади використання Python

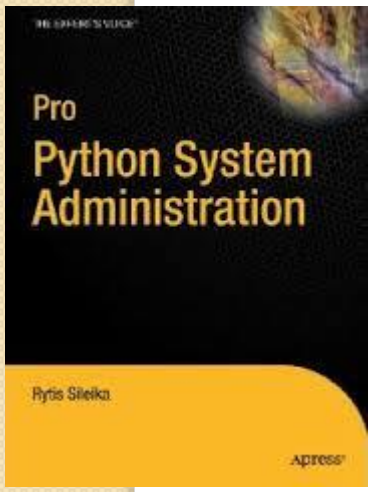
- Компанії **JPMorgan Chase, UBS, Getco і Citadel** застосовують Python для **прогнозування фінансового ринку**
- Популярна програма **BitTorrent** для **обміну файлами** в пірінгових мережах написана мовою Python
- Популярний **веб-фреймворк App Engine** від компанії Google використовує Python в якості прикладного мови програмування
- **NASA, Los Alamos, JPL і Fermilab** використовують Python для **наукових обчислень**
- The **Civilization IV game's** customizable scripted events are written entirely in Python.
- For traditional **database** demands, there are Python interfaces to all commonly used relational database systems — **Sybase, Oracle, Informix, ODBC, MySQL, PostgreSQL, SQLite**, and more.

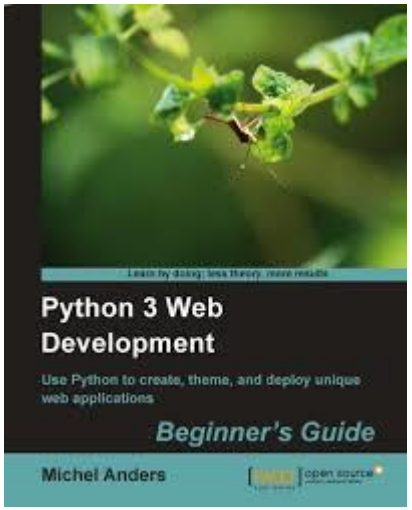

Python for Data Analysis
O'REILLY
Wes McKinney


Parallel Programming with Python
Jan Palach
PACKT

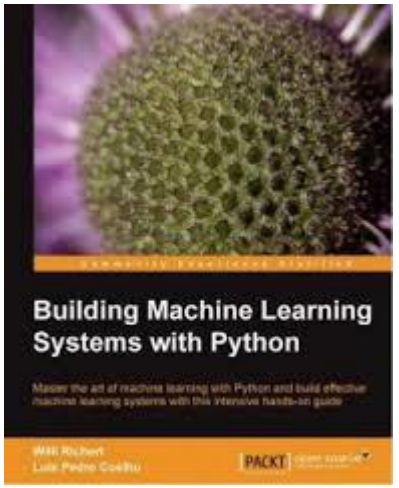

Brain-Friendly
Python For Linguists
A Hands-On Tutorial
DRAFT
Ron Zacharski


Programming ArcGIS 10.1 with Python Cookbook
Eric Pimpler


Pro
Python System Administration
Rytis Sileika
Apress


Python 3 Web Development
Use Python to create, theme, and deploy unique web applications
Beginner's Guide
Michel Anders
open source


Hans Petter Langtangen
A Primer on Scientific Programming with Python
Springer


Building Machine Learning Systems with Python
PACKT

Дзен мови Python

- В основі мови лежать кілька коротких **принципів конструювання програм**, іменованих "дзен мови Python".
- Їх текст виводиться на екран інтерпретатором, якщо набрати команду **import this**.
- Один з найбільш обговорюваних принципів наступний:

"Повинен існувати один - і, бажано, тільки один - очевидний спосіб зробити це".

Код, написаний відповідно до цього "очевидним" способом (і, можливо, не зовсім очевидним для новачка) часто характеризується як "Python'овський« (**Pythonic**).

Дзен мови Python

Якщо інтерпретаторові Пайтона дати команду import this (імпортувати "сам об'єкт"), то виведеться так званий "Дзен Пайтона", який **ілюструє ідеологію** і особливості даної мови.

Beautiful is better than ugly. Красиве краще потворного.

- Explicit is better than implicit. Явне краще неявного.
- Simple is better than complex. Просте краще складного.
- Complex is better than complicated. Складне краще ускладненого.
- Flat is better than nested. Плоске краще вкладеного.
- Sparse is better than dense. Розріджене краще щільного.
- Readability counts. Удобочитаємість важлива.
- Special cases aren't special enough to break the rules. Приватні випадки не настільки істотні, щоб порушувати правила.
- . . . .

Переваги Пайтона

- **чистий синтаксис** (для виділення блоків слід використовувати відступи);
- **переносимість** програм (що властиве більшості інтерпретованих мов);
- стандартний дистрибутив має велику кількість корисних **модулів** (включно з модулем для розробки графічного інтерфейсу);
- можливість використання Python в **діалоговому режимі**;
- стандартний дистрибутив має просте **середовище розробки**, яке зветься **IDLE** і яке написане на мові Python;
- зручний для розв'язання математичних проблем (засоби роботи з **комплексними числами**, цілі числа довільної довжини)
- **лаконічність** (програми в 2-3 рази коротші, ніж C++)

Кращі книги

1. Mark Lutz Learning Python, Fifth Edition, 2013
2. Лутц М. Изучаем Python, том 1,2 5-е изд. — 2019, 2020
3. **Bill Lubanovic Introducing Python**. – 2020
4. Tony Gaddis Starting Out with Python, 5th Ed., 2021
5. Яковенко А.В. Основи програмування. Python. – Київ : КПІ ім. Ігоря Сікорського, 2018
6. **Прохоренок Н.А., Дронов В.А. - Python 3. Самое необходимое** – 2019
7. Прохоренок Н.А., Дронов В.А. - Python 3 и PyQt 5. Разработка приложений – 2019
8. Python Programming. A Practical Approach - 2021

Корисні посилання

1. Офіційний сайт https://www.python.org/
2. The Python Tutorial https://docs.python.org/3/tutorial/index.html
3. The Python Standard Library https://docs.python.org/3.9/library/
4. Tkinter — Python interface to Tcl/Tk https://docs.python.org/3/library/tkinter.html
5. 2020 Developer Survey https://insights.stackoverflow.com/survey/2020
6. Python Mind Map https://www.mindmeister.com/752422209/python

https://www.mindmeister.com/1543427311/python-programming

7. VISUALIZE CODE EXECUTION Learn Python, Java, C, C++, JavaScript, and Ruby http://pythontutor.com/
8. Google COLAB - Welcome To Colaboratory https://colab.research.google.com/notebooks/intro.ipynb https://colab.research.google.com/notebooks/welcome.ipynb?hl=uk
9. ANACONDA https://www.anaconda.com/products/individual
10. NumPy quickstart https://numpy.org/doc/stable/user/quickstart.html
11. Online Code Editor https://pynative.com/online-python-code-editor-to-execute-python-code/

Інтерпретатори Python

- **CPython**: базовий; написаний на мові C
- **Jython** OR JPython: Корисний для платформи Java
- **IronPython**: Корисний для додатків C#.Net
- **PyPy**: написаний на Python
- **Ruby Python**: Корисний для додатків на основі Ruby
- **Anaconda Python**: Корисний для керування обробкою даних.

Установка інтерпретатора

- Завантажити Python для будь-якої платформи можна на сайті: https://www.python.org
- Актуальна версія – Python 3.9.7 (вересень 2021 р.)

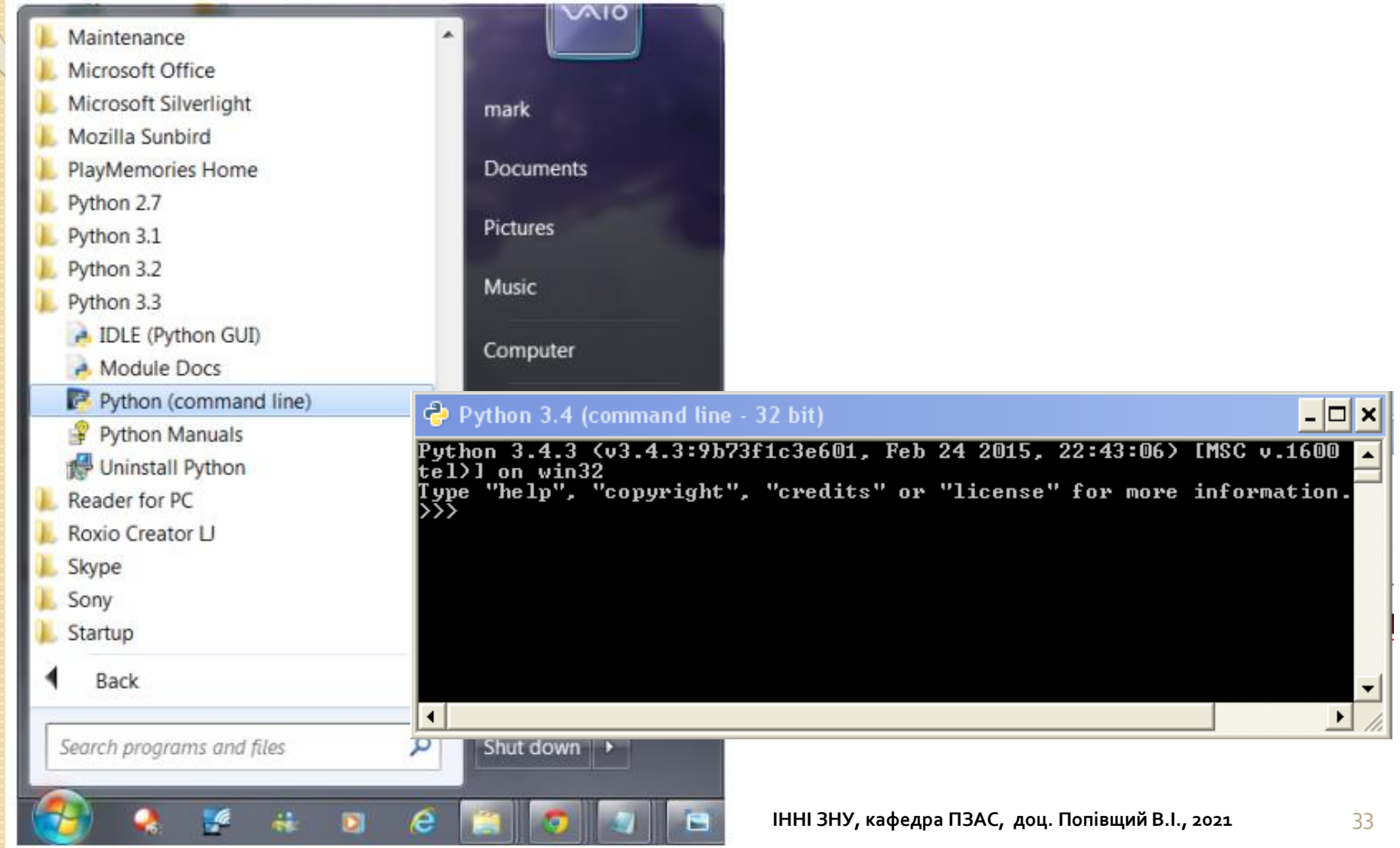
Python IDE

(Integrated Development Environment)

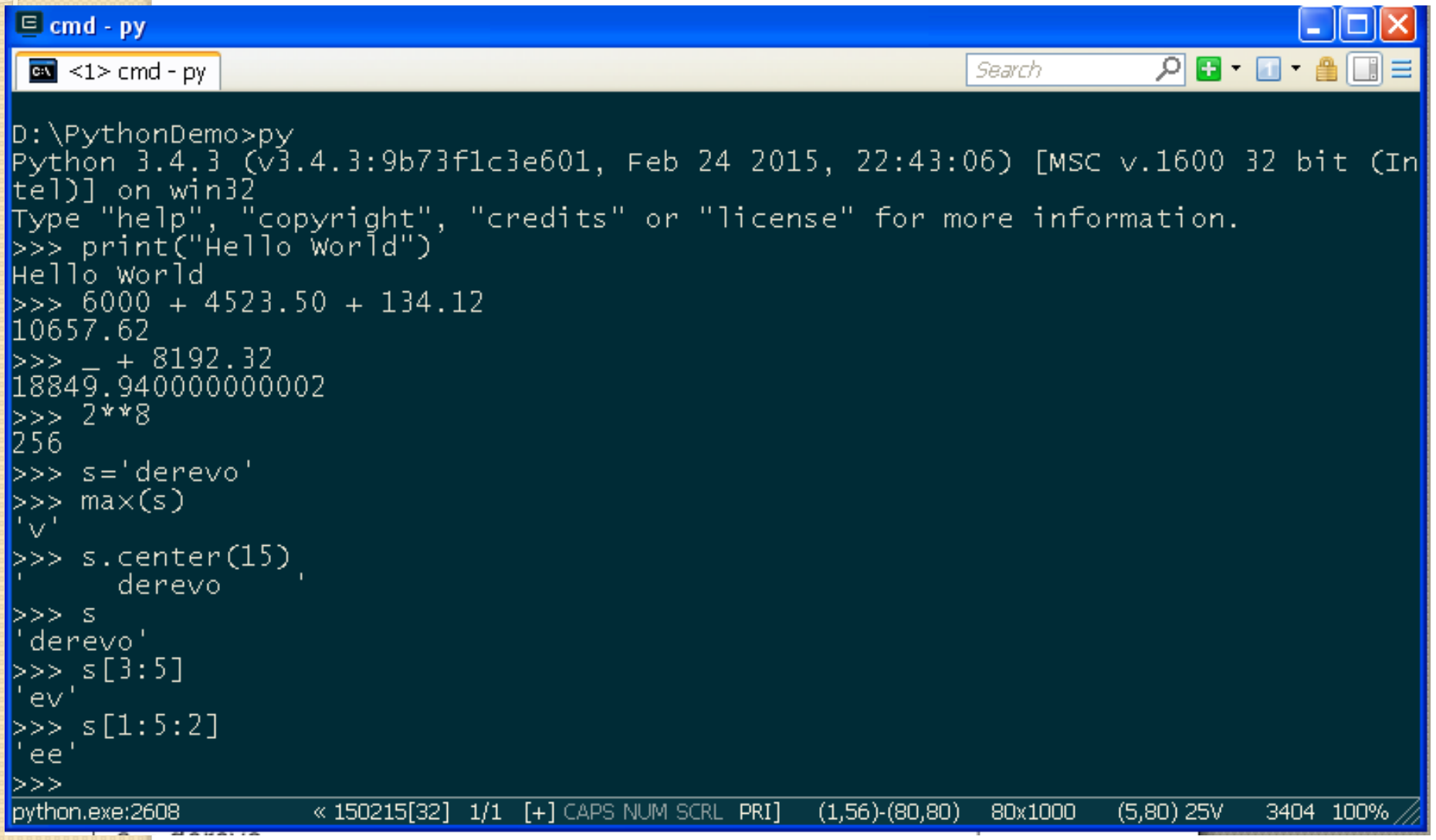
- **IDLE** - поставляється з компілятором
- **PyCharm** (https://www.jetbrains.com/pycharm/)
- **IPython**
- **Jupyter NoteBook**
- **Visual Studio Code** (https://code.visualstudio.com/)
- **PyScripter** (https://sourceforge.net/projects/pyscripter/)

Starting an Interactive Session

- Після інсталяції Python можна запустити інтерактивну консоль, вибравши в меню Пуск Python 3.4 (Command Line)



Інтерактивна консоль (Python Shell)



Відступи в коді

- Функції в Pyton не мають явних операторів begin/end або фігурних дужок для позначення початку й кінця коду функції.
- Єдиним розділювачем є двокрапка (:) та відступи в коді
- Блоки коду описуються їх відступами.
- «Блок коду" – це функції, умовні оператори, тіла циклів і так далі.
- Відступ позначає початок блоку, а прибирання відступу - його закінчення.
- Явні дужки чи ключові слова не потрібні

Стандарт PEP 8

PEP (python enhanced proposal - заявки на поліпшення мови python)

Цей стандарт – посібник з написання коду на Python

- **Використовуйте 4 пробіли на кожен рівень відступу**
- **Python 3 забороняє змішування табуляції і пробілів в відступах**
- foo = long_function_name(var_one, var_two,
  var_three, var_four)

  def long_function_name(var_one, var_two, var_three, var_four):
  print(var_one)
- **Обмежте довжину рядка максимум 79 символами**
- Відокремлюйте функції верхнього рівня і визначення класів **двома порожніми рядками**
- **Кодування Python 3 повинно бути UTF-8** (ASCII в Python 2)

Стандарт PEP 8

Слід уникати використання пробілів в наступних ситуаціях:

- **Усередині круглих, квадратних і фігурних дужок**

  animals(tiger[2], {eagle: 6})

- **Перед комою, крапкою з комою і двокрапкою**

  if x == y:

- **Відразу перед відкриваючою дужкою, після якої слідує індекс** - dict['key'] = list[index]

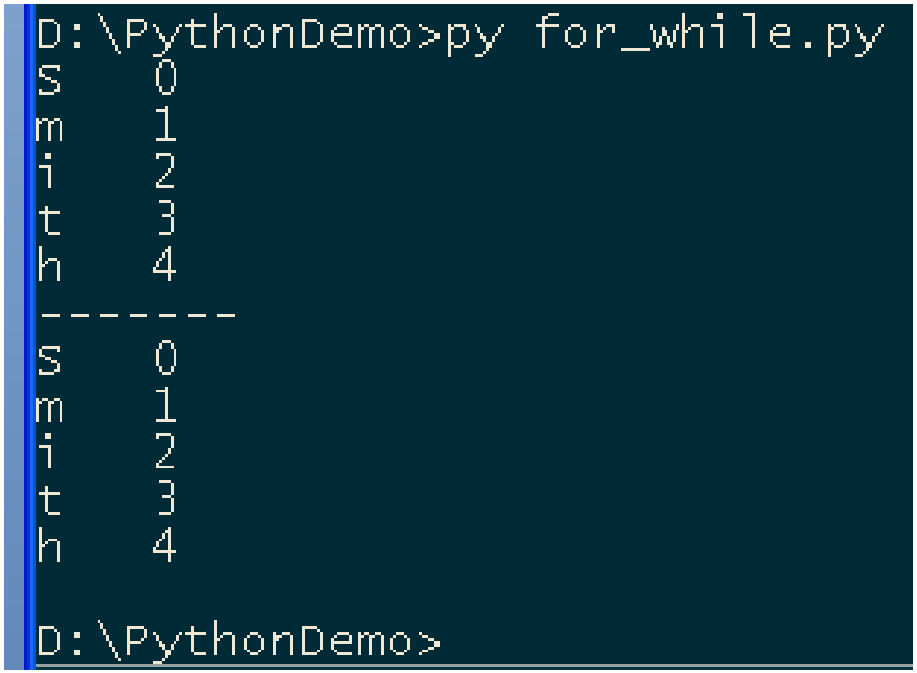
- **Завжди оточуйте наступні бінарні оператори одним пропуском з кожного боку**: оператори присвоювання (=, + =, - = і інші), оператори порівняння (==, <,>,! =, <>, <=,> =, In, not in, is, is not), логічні оператори (and, or, not)

- **Не використовуйте пробіли навколо знаку =,** якщо він використовується для позначення іменованого аргументу або повернуться до стандартних значень

Виконання файлу

Файл for_while.py

```
lastName="Smith"
count=0
for letter in lastName:
    print(letter," ",count)
    count+=1
print("-------")
count=0
while (count<5):
    print(lastName[count]," ",count)
    count+=1
```



Виконання файлу

Файл count_lines.py

```
def count_lines(filename):
    """
    Count the number of lines in a file. If the file can't be
    opened, it should be treated the same as if it was empty.
    """
    try:
        return len(open(filename, 'r').readlines())
    except IOError:
        # Something went wrong reading the file.
        return 0
myfile=input("Enter a file to open:  ")
print(count_lines(myfile))
```

```
D:\PythonDemo>py count_lines.py
Enter a file to open:  count_lines.py
12
```

Введення даних

```
>> age = input(“Please, enter your age :  “)
Please, enter your age :  24
>> print( type(age))
<class ‘str’>
>> print (“You age is   “, age)
You age is  24
```

Файл sum_int.py

```
n1 = int(input(“Enter n1 : “))
n2 = int(input(“Enter n2 : “))
sum = n1 + n2
print(“n1 + n2 = “, sum)
```

Замість int можна float

Ключові слова Python

and	del	from	not	while
as	elif	global	or	with
assert	else	if	pass	yield
break	except	import	print	
class	exec	in	raise	
continue	finally	is	return	
def	for	lambda	try	
None	False		True	

Конструкція if

```
def sign_of_a(a):
    if  a < 0.0:
            sign = ‘negative’
    elif  a > 0.0:
            sign = ‘positive’
    else:
            sign = ‘zero’
    return sign

a = 1.5
print( ‘a  is  ‘ + sign_of_a(a))
```

Result:

a is positive

Конструкція while

```
nMax = 5
n = 1
a = [ ]
while n < nMax:
    a.append(1.0/n)
    n = n + 1
print(a)
```

Результат:

[1.0, 0.5, 0.3333333333333331, 0.25]

Конструкція for

```
nMax = 5
a = [ ]
for n in range(1, nMax):
    a.append(1.0/n)
print(a)
```

Результат:

[1.0, 0.5, 0.3333333333333331, 0.25]

Python's Core Data Types

Object type	Example literals/creation
Numbers	`1234, 3.1415, 3+4j, 0b111, Decimal(), Fraction()`
Strings	`'spam', "Bob's", b'a\x01c', u'sp\xc4m'`
Lists	`[1, [2, 'three'], 4.5], list(range(10))`
Dictionaries	`{'food': 'spam', 'taste': 'yum'}, dict(hours=10)`
Tuples	`(1, 'spam', 4, 'U'), tuple('spam'), namedtuple`
Files	`open('eggs.txt'), open(r'C:\ham.bin', 'wb')`
Sets	`set('abc'), {'a', 'b', 'c'}`
Other core types	Booleans, types, None
Program unit types	Functions, modules, classes (Part IV, Part V, Part VI)
Implementation-related types	Compiled code, stack tracebacks (Part IV, Part VII)

Карта типів даних

Числа:

- цілі
- звичайне ціле **int**
- ціле довільній точності **long**
- логічний **bool**
- число з плаваючою точкою **float**
- комплексне число **complex**

Послідовності:

- незмінні:
  - рядок **str**;
  - Unicode-рядок **unicode**;
  - кортеж **tuple**;
- змінні:
  - список **list**;

відображення:

- словник **dict**

множини;

- **set**

модулі;
файли file;

об'єкти, які можна викликати:

- *функції* (користувальницькі і вбудовані);
- *функції-генератори*;
- *корутіни*;
- *методи* (користувальницькі і вбудовані);
- *класи*;
- *екземпляри* класів

Операції з числами

- **X + Y, X – Y , X * Y , X / Y**
- **X // Y** Цілочисленне ділення (результат – ціле)
- **X % Y** Залишок від цілочисленого ділення
- **X ** Y** Піднесення до степеня (x в степені y)

- Крім того, в Python для операцій з числами використовують функції **abs**(), **pow**(), **divmod**(17,5) -> (3,2), **round**()

- Це “вбудовані” функції

- Для інших математичних функцій треба підключити модуль **math**

```
>>> import math
>>> math.sqrt(85)
9.219544457292887
```

Рядки

```
>>> S = ‘Spam’
>>> len(S)
4
>>> S[0]
‘S’
>>> S[-1]
‘m’
>>> S[-2]
‘a’
>>> S[1:3]    # Slice – “зріз”
‘pa’
>>> S + ‘xyz’    # Concatenation
‘Spamxyz’
>>> S             # S is unchanged
‘Spam’
>>> S * 8         # Repetition
‘SpamSpamSpamSpamSpamSpamSpamSpam’
```

Рядки

```
>>> S = ‘Spam’
>>> S.find(‘pa’)
1
>>> S.replace('pa', 'XYZ')
'SXYZm'
>>> S
'Spam‘

>>> S = 'shrubbery'
>>> L = list(S)              # Expand to a list: [...]
>>> L
['s', 'h', 'r', 'u', 'b', 'b', 'e', 'r', 'y']
>>> L[1] = 'c'               # Change it in place
>>> ''.join(L)               # Join with empty delimiter
'scrubbery‘

>>> x = 3.4
>>> str(x)
‘3.4’
>>> format(x, “0.5f”)
‘3.40000’
```

Рядки

```
>>> smiles = "C(=N)(N)N.C(=O)(O)O“
>>> smiles.split(".")
['C(=N)(N)N', 'C(=O)(O)O']
```

```
if "Br" in “Brother”:
    print ("contains brother“)

email_address = “clin”
if "@" not in email_address:
    email_address += "@brandeis.edu“
```

Кортежи (tuple)

Кортеж в Python - це упорядкований набір об'єктів, в який можуть одночасно входити об'єкти різних типів

Кортеж задається перерахуванням його елементів **у круглих дужках** через кому, наприклад,

t = (12 , ’b’ , 34.6 , ’derevo’)

Елементам кортежу можна відразу зіставити які-небудь змінні:

t = (x , s1 , y , s2) =(12 , ’b’ , 34.6 , ’derevo’)

Основні операції з кортежами

len(t) , t1 + t2 , t ∗ n , t[i] , t[i : j :k] , min(t) , max(t)

```
>>> s= ’amamam ’
>>> t = tuple( s )
t → ( ’ a ’ , ’m’ , ’ a ’ , ’m’ , ’ a ’ , ’m’ )
```

Списки (Lists)

Список в Python - це упорядкований набір об'єктів, в список можуть одночасно входити об'єкти різних типів

lst = [12 , ’b’ , 34.6 , ’derevo’]

Основні операції зі списками

len(lst) , lst1 + lst2 , lst * n , lst[i] , lst[i : j :k] ,
min(lst) , max(lst) , lst [i]=x , del lst [i] , lst [i : j]=x

Основні методи списків

Додавання елементу lst .append(x)

Додавання кортежу або списку lst .extend(t)

lst .count(x) , lst .index(x) , lst .remove(x) , lst .pop(i) ,

lst .insert(i ,x) , lst.sort() , lst.reverse()

Функції zip(), list() та map()

Функція zip () дозволяє отримати з елементів різних списків список кортежів, що складаються з відповідних елементів списків

```
lst1 = [ 1 , 2 , 3 , 4 ]
lst2 = [ 'tri' , 'dva' , 'raz' ]
lst = list(zip( lst1, lst2))
lst → [ (1, 'tri ') , ( 2, 'dva' ), ( 3, 'raz' ) ]
```

Функція map () використовується для застосування однієї і тієї ж операції до елементів одного або декількох списків або кортежів

```
lst1 = [ 1 , 2 , 3 , 4 ]
lst = list(map( lambda  x :   x * 2 , lst1))
lst → [ 2 ,   4 ,   6 ,   8 ]

t1 =(1 ,2 ,3)
t2 = ( 5.0 , 6.0 , 7.0 )
t = list(map( lambda  x, y :   x/y, t1, t2))
t → [0.2000000000000001 ,   0.3333333333333331 ,
0.4285714285714285]
```

Функції list() та range()

Для перетворення рядка або кортежу в список використовується функція list ()

s= ’amamam ’
lst= list(s)
lst → [’a’ , ’m’ , ’a’ , ’m’ , ’a’ , ’m’]
t =(5 , 12 ,– 3, 7)
lst2= list(t)
ls 2 → [5 , 12 , – 3, 7]

Функція range () створює список як числову арифметичну прогресію

range (0,15, 3) → [0 , 3 , 6 , 9 , 12]

range (5) → [0, 1, 2, 3, 4]

range (2, 5) → [2, 3, 4]

Вкладені списки

```
>>> M = [[1, 2, 3],     # Матриця 3 x 3
         [4, 5, 6],
         [7, 8, 9]]
>>> M
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> M[1]    # Отримати рядок 2
[4, 5, 6]
>>> M[1][2]  # Отримати рядок 2, а потім елемент 3
6
```

Генератори списків

- Є прикладом так званого “синтаксичного цукру”, тобто конструкції, без якої легко можна обійтись, але з нею набагато краще.

- Приклад 1.

```
>>> res = [x for x in range(1, 25, 2)]
>>> print (res)
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23]
```

- Приклад 2

```
>>> res = [x**2 for x in range(1, 25, 2) if x % 3 != 0]
>>> print (res)
[1, 25, 49, 121, 169, 289, 361, 529]
```

Приклад 3

Є лог-файл, в якому зберігається статистика запитів до сервера у вигляді «ip bytes». Необхідно обчислити сумарний об'єм трафіка на кожний хост і видати його у вигляді списку в порядку убування трафіку.

```
127.0.0.1  120
10.1.1.1   210
127.0.0.1  80
```

```
raw = [x.split(" ") for x in open("log.txt")]
rmp = {}
for ip, traffic in raw:
    if ip in rmp:
        rmp[ip] += int(traffic)
    else:
        rmp[ip] = int(traffic)
lst = list(rmp.items())
lst.sort(key = lambda x: x[1])
print ("\n".join([“%s\t%d" % (host, traff) for host, traff in lst]))
```

Набори (sets)

Набір використовується для зберігання невпорядкованого набору об'єктів. Щоб створити набір, використовуйте функцію set(), або фігурні дужки { }

```
>>> s = set( [3, 9, 2, 15])
>>> t = set ( “Hello”)
>>> t
{ ‘e’, ‘l’, ‘H’, ‘o’}
>>> {1, 2, 3, 4, 5}
```

Операції над наборами

```
a = t | s     # Union of t and s
b = t & s     # Intersection of t and s
c = t – s     # Set difference (items in t, but not in s)
d = t ^ s     # Symmetric difference (items in t or s, but not both)
```

Набори (sets)

```
t.add('x')              # Add a single item
s.update([10,37,42])    # Adds multiple items to s
t.remove('H')
```

Словники (Dictionaries)

Словники в Python - невпорядковані колекції довільних об'єктів з доступом по ключу. Їх іноді ще називають асоціативними масивами або хеш-таблицями

```
>>> d = {}
>>> d
{}
>>> d = {'dict': 1, 'dictionary': 2}
>>> d
 {'dict': 1, 'dictionary': 2}
>>> d = dict(short='dict', long='dictionary')
>>> d
{'short': 'dict', 'long': 'dictionary'}
>>> d = dict([(1, 1), (2, 4)])
>>> d
{1: 1, 2: 4}
```

Словники (Dictionaries)

```
>>> D = { ‘a’: 1, ‘b’: 2, ‘c’: 3}
>>> D.keys()
   dict_keys([‘b’, ‘a’, ‘c’])
>>> D.values()
   dict_values([2, 1, 3])
>>> D.items()
   dict_items([(‘b’,2), (‘a’,1), (‘c’,3)])
>> D.copy()
{‘a’: 1, ‘b’: 2, ‘c’: 3}
```

D.clear() - очистка словника
D.update(D2)
D.get(K)
D.pop(K)

Приклад

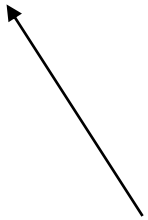
translatedictionary.py

```
translator = {'uno': 'one',
              'dos': 'two',
              'tres': 'three',
              'cuatro': 'four',
              'cinco': 'five',
              'seis': 'six',
              'siete': 'seven',
              'ocho': 'eight'}
word = '*'
while word != '':
    word = input('Enter Spanish word:')
    if word in translator:
        print(translator[word])
    else:
        print('???')    # Unknown word
```

Файли

```
f = open('data.dat')  #  f is a file object
for line in f:        #  Read each line as text
   print(line.strip())#  Remove trailing newline character
f.close()             #  Close the file
```

```
input_file = open(“in.txt")
output_file = open(“out.txt", "w")
for line in input_file:
    output_file.write(line)
```



“**w**” = “write mode”
“**a**” = “append mode”
“**wb**” = “write in binary”
“**r**” = “read mode” (default)
“**rb**” = “read in binary”
“**U**” = “read files with Unix or Windows line endings”

Приклад

wordcount.py

```
''' Uses a dictionary to count the number of occurrences of each word in
    a text file. '''
import sys    # For sys.argv global command line arguments list
def main():
    ''' Counts the words in a text file. '''
    if len(sys.argv) < 2:    # The use supply a file name?
        print('Usage: python wordcount <filename> ')
        print('  where <filename> is the name of a text file.')
    else:    # User provided file name
        filename = sys.argv[1]
        counters = {}  # Initialize counting dictionary
        with open(filename, 'r') as f
            content = f.read()
            words = content.split() # Make list of individual words
```

```
words = content.split()
for word in words:
    word = word.upper()  # Make the word all caps
    if word not in counters:
        counters[word] = 1    # First occurrence
    else:
        counters[word] += 1  # Increment existing counter
# Report the counts for each word
for word, count in counters.items():
    print(word, count)
if __name__ == '__main__':
    main()
```

Функції

Функцію можна визначити двома способами :

- за допомогою ключового слова def
- *lambda*-виразом

```
def  remainder (a, b):
    q = a // b        # // is truncating division.
    r = a - q*b
    return r
```

Виклик функції : result = remainder(37,15)

Змінні, визначені всередині функції, є локальними

```
count = 0
...
def foo():
    global count
    count += 1          # Changes the global variable count
```

Приклад

```
def multi_table(a):
    for i in range(1, 11):
        print('{0} x {1} = {2}'.format(a, i, a*i))
if __name__ == '__main__':
    a = input('Enter a number: ')
    multi_table(float(a))
```

```
Enter a number: 5
5.0 x 1 = 5.0
5.0 x 2 = 10.0
5.0 x 3 = 15.0
5.0 x 4 = 20.0
5.0 x 5 = 25.0
5.0 x 6 = 30.0
5.0 x 7 = 35.0
5.0 x 8 = 40.0
5.0 x 9 = 45.0
5.0 x 10 = 50.0
```

Функції

Можна використовувати кортежі, щоб повернути декілька значень

```
def divide(a,b):
    q = a // b          # If a and b are integers, q is integer
    r = a - q*b
    return (q,r)
```

Виклик : q, r = divide(1456, 33)

Можна задавати значення аргументів за замовчуванням

```
def connect(hostname, port, timeout=300):
    # Function body
```

Виклик : connect('www.python.org', 80)

або так - connect(port=80,hostname="www.python.org")

Генератори

Замість повернення одиночного значення, функція може згенерувати послідовність результатів, якщо вона використовує ключове слово **yield**. Наприклад:

```
def countdown( n):
    print  ("Counting down!“)
    while n > 0:
        yield n       # Generate a value (n)
        n -= 1
```

Кожну функцію, яка використовує yield, називають генератором

```
>>> c = countdown(5)
>>> next(c)
Counting down!
5
>>> next(c)
4
```

```
>>> for  i  in  countdown(5):
...     print  (i, “ “)
Counting down!
5  4  3  2  1
```

Співпрограми (корутіни, coroutines)

Як правило, функції працюють на одному наборі вхідних аргументів. Однак, можна написати **функцю, яка обробляє послідовність вхідних відправлених їй даних**.

Цей тип функцій, що називають співпрограмами , можна створити за допомогою інструкції **yield** у вигляді **виразу**, як показано на прикладі :

```
def print_matches(matchtext):
    print "Looking for", matchtext
    while True:
        line = (yield)  # Get a line of text
        if matchtext in line:
            print line
```

Використання:

```
>>> matcher = print_matches("python")
>>> matcher.next()  # Advance to the first (yield)
Looking for python
>>> matcher.send("Hello World")
>>> matcher.send("python is cool")
python is cool
>>> matcher.send("yow!")
>>> matcher.close()    # Done with the matcher function call
```

Співпрограми

Напишемо просту реалізацію генератора, який може додавати два аргументи, зберігати історію результатів і виводити історію (http://habrahabr.ru/post/196918/).

```
def calc():
    history = []
    while True:
        x, y = (yield)
        if x == 'h':
            print history
            continue
    result = x + y
    print result
    history.append(result)
```

Використання:

```
c = calc()
print type(c) # <type 'generator'>
c.next()
c.send((1,2)) # Виведе 3
c.send((100, 30)) # Виведе 130
c.send((666, 0)) # Виведет 666
c.send(('h',0)) # Виведе [3, 130, 666]
c.close() # Закриваємо генератор
```

Декоратори (1)

- Декоратори в мові Python являють собою механізм, що дозволяє змінювати, “декорувати” поведінку функцій, примушуючи їх працювати дещо інакше
- Декоратор – це “обгортка навколо функції”
- Синтаксис застосування декораторів:

```
@deco
def  foo():
      pass
```

Рівнозначно виклику
foo = deco(foo)

- deco – це функція-декоратор

Декоратори (2)

Приклад. Реєстрація кожного виклику функції

```
def  log( func) :
      def  wrappedFunc() :
            print( “*** %s() called”, func.__name__)
            return func()
      return wrappedFunc

@log
def  foo() :
      print( “inside foo()”)

>>> foo()
*** foo() called
 inside foo()
```

Модулі (1)

Модулі виконують дві важливих функції:

- Повторне використання коду
- Управління адресним простором імен

Python дозволяє помістити класи, функції або дані в окремий файл і використовувати їх в інших програмах. Такий файл називається модулем.

Підключити модуль можна за допомогою інструкції **import**

```
>>> import os
>>> os.getcwd()
'C:\\Python33‘
```

```
>>> import math
>>> math.e
2.718281828459045
```

Використання псевдонімів

```
>>> import math as m
>>> m.e
2.718281828459045
```

Модулі (2)

Інструкція from

- from <Назва модуля> import <Атрибут 1> [as <Псевдонім 1>], [<Атрибут 2> [as <Псевдонім 2>] ...]
- from <Назва модуля> import *

```
>>> from math import e, ceil as c
>>> e
2.718281828459045
>>> c(4.6)
5
```

- Можна створювати свої модулі
- Шляхи пошуку модулів вказані у змінній sys.path (поточна директорія, каталоги, де встановлений Python)

```
>>> import sys
>>> sys.path
['', 'C:\\WINDOWS\\system32\\python34.zip', 'D:\\Python34\\DLLs', 'D:\\Python34\
\lib', 'D:\\Python34', 'D:\\Python34\\lib\\site-packages']
>>>
```

Модулі (3)

```
# Fibonacci numbers module fibo.py
def fib(n): # write Fibonacci series up to n
    a, b = 0, 1
    while b < n:
        print(b, end=' ')
        a, b = b, a+b
    print()

def fib2(n): # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a+b
    return result
```

Модулі (4)

```
>>> import fibo
>>> fibo.fib(1000)
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
>>> fibo.fib2(100)
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
>>> fibo.__name__
'fibo‘

>>> fib = fibo.fib
>>> fib(500)
1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

Executing modules as scripts

When you run a Python module with

```
python fibo.py <arguments>
```

the code in the module will be executed, just as if you imported it, but with the __name__ set to "__main__". That means that by adding this code at the end of your module:

```
if __name__ == "__main__":
    import sys
    fib(int(sys.argv[1]))
```

you can make the file usable as a script as well as an importable module, because the code that parses the command line only runs if the module is executed as the “main” file:

```
$ python fibo.py 50
1 1 2 3 5 8 13 21 34
```

If the module is imported, the code is not run:

```
>>> import fibo
>>>
```

This is often used either to provide a convenient user interface to a module, or for testing purposes (running the module as a script executes a test suite)

Стандартні модулі Python

- Модуль fractions надає підтримку раціональних чисел
- Модуль cmath - надає функції для роботи з комплексними числами
- Модуль copy - поверхневе і глибоке копіювання об'єктів
- Модуль functools - збірка функцій високого рівня, взаємодіючих з іншими функціями
- Модуль os надає безліч функцій для роботи з операційною системою
- Модуль json
- Модуль pickle реалізує потужний алгоритм серіалізаціі і десеріалізаціï об'єктів
- . . . . . .

Модуль re

- Механізми для роботи з регулярними виразами доступні в Python у вигляді модуля re
- Одним з часто використовуваних компонентів модуля є функція search
- Функція re.search повертає об'єкт співпадання, що має методи group та groups, через які можна отримати співпадання

```
>>> import re
>>> m = re.search( r'foo', 'seafood')
>>> print( m)
<_sre.SRE_Match object; span(3,6), match='foo'>
>>> m.group()
'foo'
>>>
```

Модуль random

Модуль random надає функції для генерації випадкових чисел, букв, випадкового вибору елементів послідовності.

```
import random
print (random.randint(0, 5))
print(random.random() )    # Random float x, 0.0 <= x < 1.0
print (random.randrange(0, 101,2)) # Even integer from 0 to 100
print (random.uniform(0, 5))  # дійсне число
myList = [2, 109, False, 10, "Lorem", 482, "Ipsum"]
print(random.choice(myList))

>>>items = [1, 2, 3, 4, 5, 6, 7]
>>>random.shuffle(items)
>>>items
[7, 3, 2, 5, 6, 4, 1]
```

Об’єкти та класи

- Всі значення, використовувані програмами, є об’єкти
- Об’єкт містить деякі внутрішні дані та має методи, що дозволяють виконувати різні операції над цими даними
- Функція **dir ()** виводить список всіх методів об'єкта

```
>>> items = [37, 42]
>>> dir(items)
[‘__add__’, ‘__class__’, ‘__contains__’, ‘__delattr__’,
‘__delitem__’,
        ...
‘append’, ‘count’, ‘extend’, ‘index’, ‘insert’, ‘pop’,
‘remove’, ‘reverse’, ‘sort’]
>>>
```

Об'єкти та класи

Для визначення нових типів об'єктів використовується інструкція **class**

Приклад. Клас стека

```
class Stack(object):
    def __init__(self):    # Ініціалізація стека
     self.stack = [ ]
    def push(self,object):
        self.stack.append(object)
    def pop(self):
        return self.stack.pop()
    def length(self):
        return len(self.stack)
```

- Клас Stack успадковує властивості і методи класу **object**
- self – аналог this (посилання на екземпляр класу)

Об'єкти та класи

- Методи, імена яких починаються і закінчуються двома символами підкреслення, є спеціальними методами

```
s = Stack()
s.push("Dave")
s.push(42)
s.push([3,4,5])
x = s.pop()      # x отримає значення [3,4,5]
y = s.pop()      # y отримає значення 42
del s            # Знищує об'єкт s
```

Всі методи класу Stack можуть застосовуватися тільки до екземплярів даного класу (тобто до створених об'єктів)

Статичні методи

```
class EventHandler(object):
    @staticmethod
    def dispatcherThread():
        while (1):
            # Чекання запиту
...
```

```
EventHandler.dispatcherThread() # Виклик метода
```

- В даному випадку @staticmethod оголошує наступний за ним метод статичним.
- @staticmethod - це приклад використання *декоратора*

Приклад класу

```
class Car:
    def __init__( self, make, model, year)
            self.make = make
            self.model = model
            self.year = year
            print("Instance object of the class is created")

    def displayParameters(self) :
            print("Make  : ", self.make)
            print("Model  : ", self.model)
            print("Year  : ", self.year)
            print()


car1 = Car("A", "B", 2015)
car1.displayParameters()
```

Class variable

```
class Car:
    totalNumber = 0
    def __init__( self, make, model, year)
            Car.totalNumber +=1
            self.make = make
            self.model = model
            self.year = year
            print("Total number of car is ", Car.totalNumber)

    def displayParameters(self) :
            print("Make  : ", self.make)
            print("Model  : ", self.model)
            print("Year  : ", self.year)
            print()
car1 = Car("A", "B", 2015)
car2 = Car("C", "D", 2014)
```

Деструктор

```
class Car:
    totalNumber = 0
    def __init__( self, make, model, year) :
            Car.totalNumber +=1
            self.make = make
            self.model = model
            self.year = year
    def __del__ (self) :
            Car.totalNumber -=1

    def displayParameters(self) :
            print("Make  : ", self.make)
            print("Model  : ", self.model)
            print("Year  : ", self.year)
car1 = Car("A", "B", 2015)
car2 = Car("C", "D", 2014)
del car2
```

Приватні поля класу та властивості (1)

Ми можемо приховати атрибути класу (зробити їх приватними), якщо перед іменем атрибуту поставимо два підкреслення.

```
class Gadget:        # Файл Gadget.py
    """A class used for modelling Gadgets in a web shop."""
    __weight = 100
    __operating_system = None
    __battery_capacity = 2000
    __screen_size = 1
    def __init__(self, weight, operating_system, battery_capacity,
screen_size):
        self.__weight = weight
        self.__operating_system = operating_system
        self.__battery_capacity = battery_capacity
        self.__screen_size = screen_size
    def get_weight(self):
        return self.__weight
    def set_weight(self, weight):
        self.__weight = weight
```

(закінчення далі)

Приватні поля класу та властивості (2)

```
weight = property(get_weight, set_weight)
@property
def operating_system(self):
    return self.__operating_system
@operating_system.setter
def operating_system(self, new_os):
    self.__operating_system = new_os
```

Застосування

```
>>> from Gadget import Gadget
>>> my_iphone = Gadget(240,'iOS',1980,4)
>>> my_iphone.weight
240
>>> my_iphone.weight = 255
>>> my_iphone.weight
255
>>> my_iphone.operating_system
'iOS'
>>> my_iphone.operating_system = 'iOS 8.1'
>>> my_iphone.operating_system
'iOS 8.1'
```

Single Inheritance in **Python** (1)

In **Python**, inheritance can be done via the class MySubClass(MyBaseClass) syntax

```
class Animal:
    __name = None
    __age = 0
    __is_hungry = False
    __nr_of_legs = 0
    def __init__(self, name, age, is_hungry, nr_of_legs):
        self.name = name
        self.age = age
        self.is_hungry = is_hungry
        self.nr_of_legs = nr_of_legs
    def eat(self, food):
        print("{} is eating {}.".format(self.name, food))
    @property
    def name(self):
        return self.__name
# Закінчення далі
```

Single Inheritance in **Python** (2)

```
@name.setter
def name(self,new_name):
    self.__name = new_name
@property
def age(self):
    return self.__age
@age.setter
def age(self,new_age):
    self.__age = new_age
@property
def is_hungry(self):
    return self.__is_hungry
@is_hungry.setter
def is_hungry(self,new_is_hungry):
    self.__is_hungry = new_is_hungry
@property
def nr_of_legs(self):
    return self.__nr_of_legs
@nr_of_legs.setter
def nr_of_legs(self,new_nr_of_legs):
    self.__nr_of_legs = new_nr_of_legs
```

Single Inheritance in **Python** (3)

```
class Snake(Animal):
    __temperature = 28
    def __init__(self, name, temperature):
        super().__init__(name, 2, True, 0)
        self.temperature = temperature
    def eat(self, food):
        if food == "meat":
            super().eat(food)
        else:
            raise ValueError
    @property
    def temperature(self):
        return self.__temperature
    @temperature.setter
    def temperature(self,new_temperature):
        if new_temperature < 10 or new_temperature > 40:
                raise ValueError
        self.__temperature = new_temperature
```

Multiple Inheritance in **Python**

```
class Phone(object):
    def __init__(self):
        print("Phone constructor invoked.")
    def call_number(self, phone_number):
        print("Calling number {}".format(phone_number))
```

```
class Computer(object):
    def __init__(self):
        print("Computer constructor invoked.")
    def install_software(self, software):
        print("Computer installing the {} software".format(software))
```

```
class SmartPhone(Phone,Computer):
    def __init__(self):
        Phone.__init__(self)
        Computer.__init__(self)
```

. . . . . .

```
>>> my_iphone = SmartPhone()
Phone constructor invoked.
Computer constructor invoked.
>>> my_iphone.call_number("123456789")
Calling number 123456789
```

Обробка винятків (1)

```
try:
   Value1 = int(input("Type the first number: "))
   Value2 = int(input("Type the second number: "))
   Output = Value1 / Value2
except  ValueError:
   print("You must type a whole number!")
except  KeyboardInterrupt:
   print("You pressed Ctrl+C!")
except  ZeroDivisionError:
   print("Attempted to divide by zero!")
except  ArithmeticError:
   print("An undefined math error occurred.")
else:
   print(Output)
```

Обробка винятків (2)

```
TryAgain = True
while TryAgain:
  try:
    Value = int(input("Type a whole number. "))
  except  ValueError:
    print("You must type a whole number!“)
    try:
      DoOver = input("Try again (y/n)? ")
    except:
      print("OK, see you next time!")
      TryAgain = False
    else:
      if (str.upper(DoOver) == "N"):
        TryAgain = False
  except KeyboardInterrupt:
    print("You pressed Ctrl+C!")
    print("See you next time!")
    TryAgain = False
  else:
    print(Value)
    TryAgain = False
```

Генерація винятків

Приклад 1

```
try:
   raise ValueError
except ValueError:
   print("ValueError Exception!")
```

Приклад 2

```
try:
   Ex = ValueError()
   Ex.strerror = "Value must be within 1 and 10."
   raise Ex
except ValueError as e:
   print("ValueError Exception!", e.strerror)
```

Створення власного винятку

```
class CustomValueError(ValueError):
    def __init__(self, arg):
        self.strerror = arg
        self.args = {arg}
try:
    raise CustomValueError("Value must be within 1 and 10.")
except CustomValueError as e:
    print("CustomValueError Exception!", e.strerror)
```

Використання finally

```
import sys
try:
   raise ValueError
   print("Raising an exception.")
except  ValueError:
   print("ValueError Exception!")
   sys.exit()
finally:
   print("Taking care of last minute details.")
print("This code will never execute.")
```

Якщо навіть закоментувати 3-й рядок, все одно останній оператор print не буде виконаний

Пакети

Пакети в мові Python забезпечують спосіб розподілу набора модулів по каталогам файлової системи та використання точкової нотації для доступу до модулів підпакетів

```
Sound/                          Верхний уровень пакета
      __init__.py               Инициализация пакета
      Formats/                  Работа с файлами
            __init__.py
            wavread.py
            wavwrite.py
            aiffread.py
            aiffwrite.py
            auread.py
            auwrite.py
            ...
      Effects/                  Звуковые эффекты
            __init__.py
            echo.py
            surround.py
            reverse.py
            ...
      Filters/                  Фильтры
            __init__.py
            equalizer.py
            vocoder.py
            karaoke.py
            ...
```

Управління пакетами

- Пакетами Python можуть бути інструменти, бібліотеки, фреймворки, застосунки
- Існують десятки тисяч доступних пакетів, які можна використовувати для створення власних проектів

Інструменти управління пакетами

Найбільш часто використовувані менеджери пакетів Python – це **pip** і **easy_install**.

Дані інструменти допомагають виконати наступні **завдання**:

- Скачування, установка, видалення пакетів;
- Зборка пакетів;
- Управління пакетами Python і багато іншого

Python GUI Programming

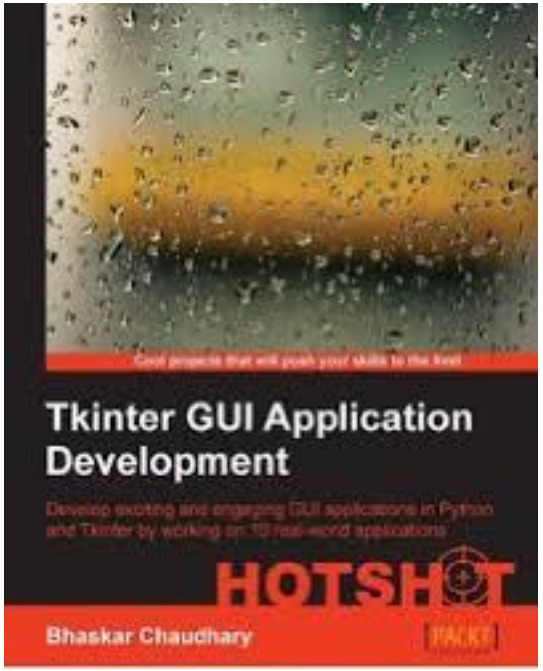
Для розробки застосунків з графічним інтерфейсом користувача можна застосовувати:

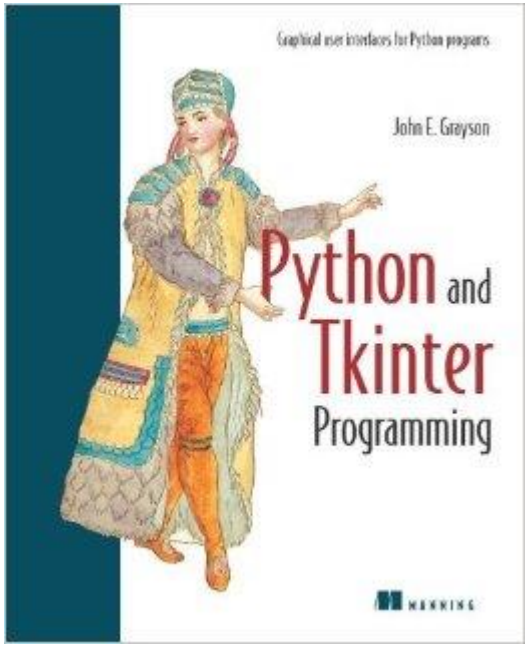
- **Tkinter** (є інтерфейсом Python до інструментарію Tk GUI, поставляється з Python)
- **wxPython** - (Це інтерфейс з відкритим вихідним кодом Python для wxWindows http://wxpython.org)
- **PyQt** (набір «прив'язок» графічного фреймворку Qt для мови програмування Python, виконаний у вигляді розширення Python. 600 класів, 6000 функцій, http://www.qt.io, http://riverbankcomputing.com)
- **Kivy** (бібліотека Python для швидкої розробки додатків з використанням інноваційних користувальницьких інтерфейсів, таких як мульти-сенсорні додатки – kivy.org)

Tkinter

Lynda.com - Python GUI Development with Tkinter, 2014

590 Mb



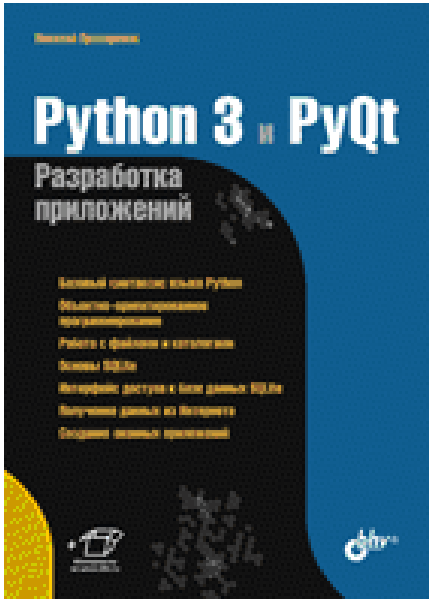


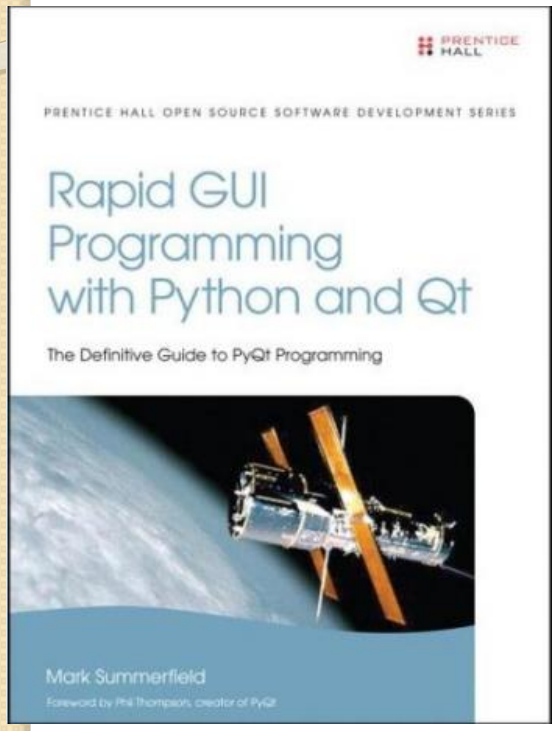
A window with a label and two buttons

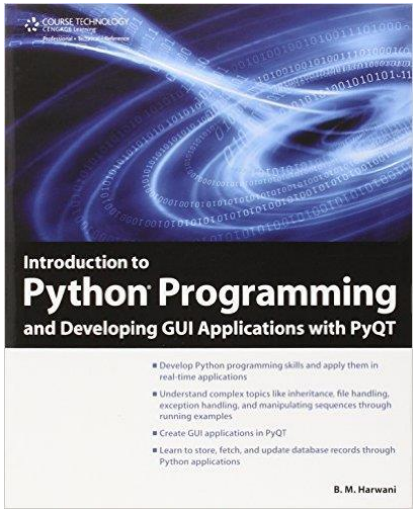
```
from tkinter import Tk, Label, Button
class MyFirstGUI:
    def __init__(self, master):
        self.master = master master.title("A simple GUI")
        self.label = Label(master, text="This is our first GUI!")
        self.label.pack()
        self.greet_button = Button(master, text="Greet",
                                   command=self.greet)
        self.greet_button.pack()
        self.close_button = Button(master, text="Close",
                                   command=master.quit)
        self.close_button.pack()
    def greet(self):
        print("Greetings!")
root = Tk()
my_gui = MyFirstGUI(root)
root.mainloop()
```

Python 3 та PyQt

- Прохоренок Н. Python 3 и PyQt. Разработка приложений, 2012







Udemy - Create simple GUI Applications with Python and Qt 2015

Установка PyQt

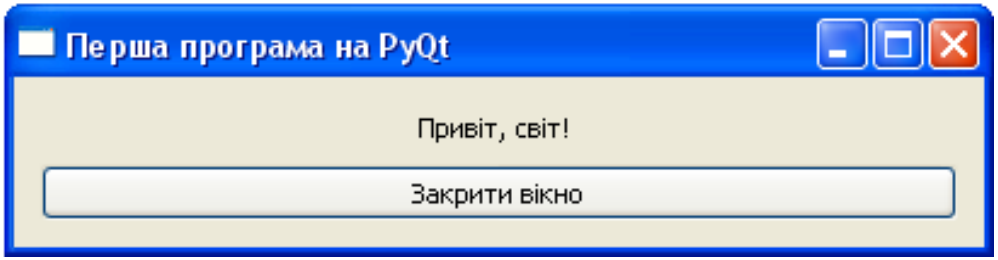
- Завантажуємо програму установки PyQt4-4.11.4-gpl-Py3.4-Qt4.8.7-x32.exe (28,4 Mb) із сторінки https://www.riverbankcomputing.com/software/pyqt/download і запускаємо її
- У результаті встановлення всі необхідні файли будуть скопійовані в папку C:\Python34\Lib\site-packages\PyQt4\, а в початок системної змінної **PATH** доданий шлях до папки C:\Python34\Lib\site-packages\PyQt4\.
- У цій папці розташовані програми- Designer, Linguist і Assistant, а також бібліотеки динамічного компонування (наприклад, QtCore4.dll, QtGui4.dll), необхідні для нормального функціонування програми

Перевірка правильності установки PyQt

```
>>>  from  PyQt4  import QtCore
>>> QtCore.PYQT_VERSION_STR
'4.11.4'
>>> QtCore.QT_VERSION_STR
'4.8.7'
>>>
```

Перша програма (1)

```
from PyQt4 import QtCore, QtGui
import sys
app = QtGui.QApplication(sys.argv)
window = QtGui.QWidget()
window.setWindowTitle("Перша програма на PyQt")
window.resize(400,70)
label = QtGui.QLabel("<center>Привіт, світ! </center>")
btnQuit = QtGui.QPushButton("&Закрити вікно")
vbox = QtGui.QVBoxLayout()
vbox.addWidget(label)
vbox.addWidget(btnQuit)
window.setLayout(vbox)
QtCore.QObject.connect(btnQuit, QtCore.SIGNAL("clicked()"), QtGui.qApp,
    QtCore.SLOT("quit()"))
window.show()
sys.exit(app.exec_())
```



Перша програма (2)

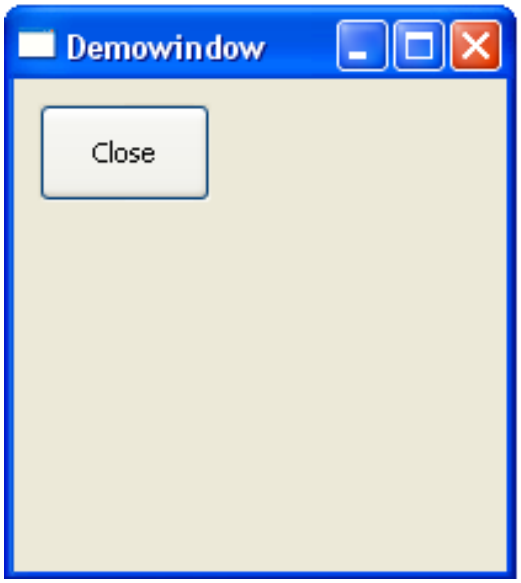
- Інструкція app = QtGui.QApplication(sys.argv)

створює об’єкт застосунку.

- Наступна інструкція window = QtGui.QWidget() створює об’єкт вікна.
- Інструкція label = QtGui.QLabel(" Привіт, світ! ") створює об’єкт напису
- Інструкція vbox = QtGui.QVBoxLayout() створює вертикальний контейнер. Усі компоненти, що додаються в контейнер, будуть розташовуват.ись один під одним.
- Інструкція QtCore.QObject.connect(btnQuit, QtCore.SIGNAL("clicked()"), QtGui.qApp, QtCore.SLOT("quit()")) призначає обробник сигналу **clicked()**, який генерується при натисканні кнопки

ООП-стиль створення вікна

```
import sys
from PyQt4 import QtGui,QtCore
class demowind(QtGui.QWidget):
    def __init__(self,parent=None):
        QtGui.QWidget.__init__(self, parent)
        self.setGeometry(300,300,200,200)
        self.setWindowTitle('Demowindow')
        quit=QtGui.QPushButton('Close',self)
        quit.setGeometry(10,10,70,40)
        self.connect(quit,QtCore.SIGNAL('clicked()'),
                     QtGui.qApp,QtCore.SLOT('quit()'))
app = QtGui.QApplication(sys.argv)
dw = demowind()
dw.show()
sys.exit(app.exec_())
```



The Tetris game in PyQt4

Для майбутнього вивчення:

http://zetcode.com/gui/pyqt4/thetetrisgame/