

# Python

## QUICK RECAP

Giovanni Squillero  
squillero@polito.it



1

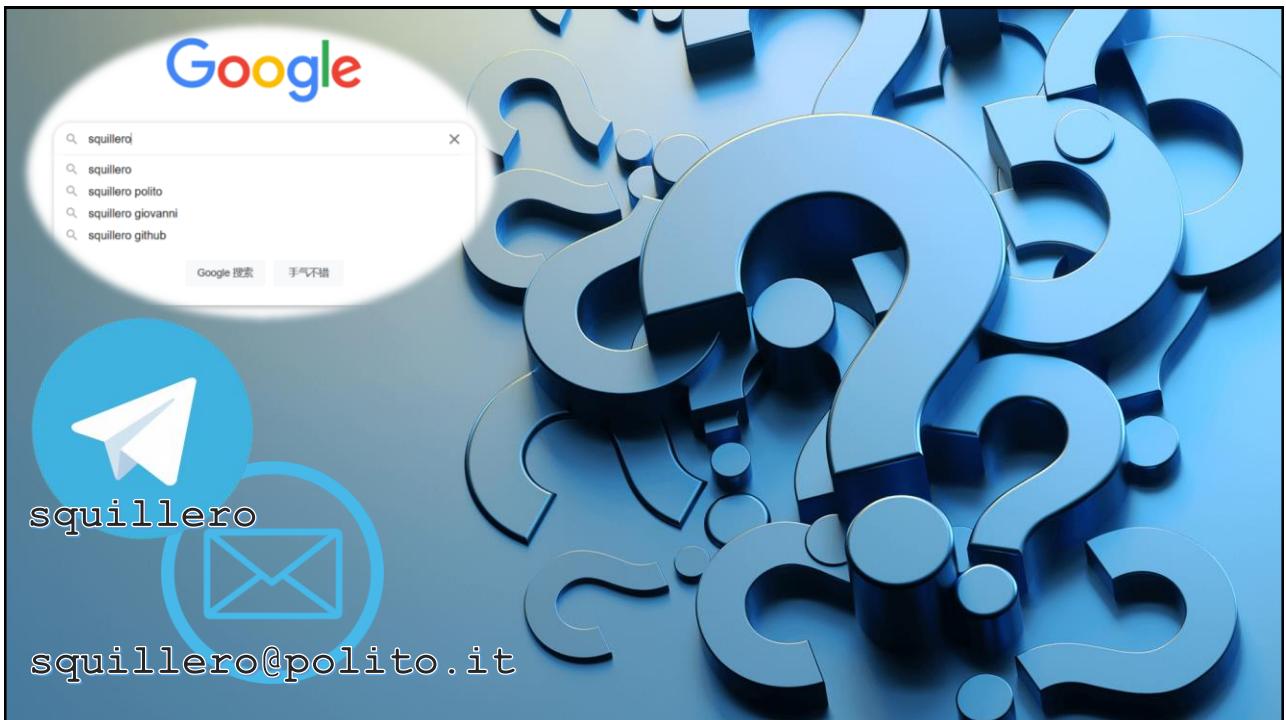
## Python quick recap

<https://github.com/squillero/10k/>

**Copyright © 2022 by Giovanni Squillero.**

Permission to make digital or hard copies for personal or classroom use of these files, either with or without modification, is granted without fee provided that copies are not distributed for profit, and that copies preserve the copyright notice and the full reference to the source repository. To republish, to post on servers, or to redistribute to lists, contact the Author. These files are offered as-is, without any warranty.

version 2022.09



3

## Why Python in 2022?

- High-level programming language, truly portable
- Actively developed, open-source, community-driven
- Batteries-included approach, huge code base
- Reasonably easy to learn and to use
- Powerful as a scripting language and glue language
- Support programming in the large
- Can be used interactively in *notebooks*
- De-facto standard in several domains (e.g., data science)

## Writing Python (scripts)

- Visual Studio Code 
- Alternatives
  - Any advanced text editor
  - Full fledged IDE

squillero@polito.it

Python Quick Recap 2022

5

5

## Writing Python (large projects)

- Free
  - PyCharm Community Edition
- Paid
  - PyCharm Professional 
- Alternatives
  - Visual Studio Code
  - Eclipse
  - Replit (online) <<https://replit.com/>>

squillero@polito.it

Python Quick Recap 2022

6

6

3

## Writing Python (notebook)

- Free
  - Just use Jupyter Notebook (or Lab)
  - Visual Studio Code
- Paid
  - PyCharm Professional
  - DataSpell 
- Online
  - Colab <<https://colab.research.google.com/>> 
  - Notebooks <<https://notebooks.ai/>>

## Running Python

- UN\*X (Linux, MacOS, ...)
  - `python foo.py` / `python3 foo.py`
  - `poetry run python foo.py`
- Windows console applications
  - `python.exe foo.py`
  - `poetry.exe run python.exe foo.py`
- Windows GUI/no-UI scripts
  - `pythonw.exe foo.py`

# Baselines

- Python modules and the language itself is evolving rapidly
- *Dependency Management* may be problematic
- Do not tamper directly with the system environment
  - **venv**: builtin module provides for creating lightweight “virtual environments” (Python’s license: OSI-approved & GPL-compatible)
  - **Poetry**: external tool for packaging and dependency management (MIT license)
  - **Anaconda**: Python distribution targeting scientific computing that includes the **conda** package manager (double check license, Anaconda free version is not compliant for commercial use)

squillero@polito.it

Python Quick Recap 2022



9

9

# Poetry

- Setup environment (or get the **pyproject.toml**)

```

giovanni@AGRITTE E:\TMP\example
$ poetry init
INFO: Could not find files for the given pattern(s).

This command will guide you through creating your pyproject.toml config.

[package]
name = "metta"
version = "0.1.0"
description = "Class example"
author = "None, or to skip: Giovanni Squillero <squillero@polito.it>"
license = "Unlicense"
compatible_python_version = ["^3.10"]

Would you like to define your main dependencies interactively? (yes/no) [yes] no
Would you like to define your development dependencies interactively? (yes/no) [yes] no
Generated file.

[tool.poetry]
name = "metta"
version = "1.0"
description = "Class example"
authors = ["Giovanni Squillero <squillero@polito.it>"]
license = "Unlicense"

[tool.poetry.dependencies]
python = "^3.10"

[tool.poetry.dev-dependencies]
[build-system]
requires = ["poetry-core>=1.0.0"]
build-backend = "poetry.core.masonry.api"

Do you confirm generation? (yes/no) [yes]

```

squillero@polito.it

Python Quick Recap 2022

10

10

# Poetry

- Install dependencies and spawn a shell

```

giova@MAGRITTE E:\TMP\example
$ poetry install
Creating virtualenv mettā-hTb4_Wcd-py3.10 in C:\Users\giova\AppData\Local\pypoetry\Cache\virtualenvs
Updating dependencies
Resolving dependencies... (0.1s) ↑

Writing lock file

giova@MAGRITTE E:\TMP\example
$ poetry shell
Spawning shell within C:\Users\giova\AppData\Local\pypoetry\Cache\virtualenvs\mettā-hTb4_Wcd-py3.10
Microsoft Windows [Version 10.0.22000.856]
(c) Microsoft Corporation. All rights reserved.

giova@MAGRITTE E:\TMP\example
$ where python
C:\Users\giova\AppData\Local\pypoetry\Cache\virtualenvs\mettā-hTb4_Wcd-py3.10\Scripts\python.exe →
C:\Program Files\Python310\python.exe
C:\msys64\mingw64\bin\python.exe

```

squillero@polito.it

Python Quick Recap 2022

11

11

# Poetry

- **install / update**
  - Installs/Update the project dependencies
- **add / remove**
  - Adds/Removes a new dependency to pyproject.toml
- **init**
  - Creates a basic pyproject.toml file in the current directory
- **shell / run**
  - Spawns a shell/Runs a single command in the virtual environment
- **show [--tree]**
  - Shows information about packages

etc.  
etc.

squillero@polito.it

Python Quick Recap 2022

12

## Poetry

- Possible problem (over SSH)

```
(10k-py3.10) > $ poetry install
Updating dependencies
Resolving dependencies... (0.0s)

Failed to create the collection: Prompt dismissed..
```

- ... and possible solution:

```
PYTHON_KEYRING_BACKEND="keyring.backends.null.Keyring"
```

squillero@polito.it

Python Quick Recap 2022

13

13

## Interactive Notebook

- Let's test Python by calculating the 17<sup>th</sup> Mersenne prime and display it with nice thousands separator...

```
f"{2**2281-1:,}"
```

```
'446,087,557,183,758,429,571,151,706,402,101,809,886,208,632,412,859,901,111,991,    :
219,963,404,685,792,820,473,369,112,545,269,003,989,026,153,245,931,124,316,702,395,
758,705,693,679,364,790,903,497,461,147,071,065,254,193,353,938,124,978,226,307,947,
312,410,798,874,869,040,070,279,328,428,810,311,754,844,108,094,878,252,494,866,760,
969,586,998,128,982,645,877,596,028,979,171,536,962,503,068,429,617,331,702,184,750,
324,583,009,171,832,104,916,050,157,628,886,606,372,145,501,702,225,925,125,224,076,
829,605,427,173,573,964,812,995,250,569,412,480,720,738,476,855,293,681,666,712,844,
831,190,877,620,606,786,663,862,190,240,118,570,736,831,901,886,479,225,810,414,714,
078,935,386,562,497,968,178,729,127,629,594,924,411,960,961,386,713,946,279,899,275,
006,954,917,139,758,796,061,223,803,393,537,381,034,666,494,402,951,052,059,047,968,
693,255,388,647,930,440,925,104,186,817,009,640,171,764,133,172,418,132,836,351'
```

squillero@polito.it

Python Quick Recap 2022

14

14

## Python — Quick Recap

# Data Types



15

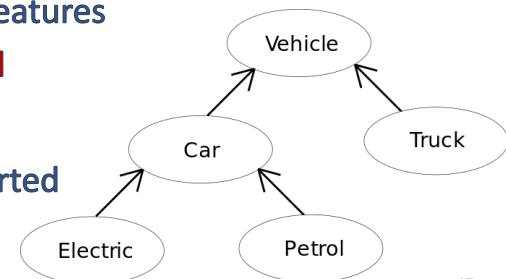
## Data Model

- Python is a **strongly-typed, dynamic, object-oriented** language
  - Objects are Python's abstraction for data
  - Code is also represented by objects
  - Every object has an **identity**
    - The identity never changes once it has been created — `is`, `id()`
  - Every object has a **type** and a **value**

# Object Oriented Paradigm

- An **object** contains both **data** and **code**
- An **object** is the instance of a **class** (class ↔ type)
- Subclass hierarchy
  - A class **inherits** the structure from its parent(s)
  - The child class may **add** or **specialize** features
  - **isinstance(x, Car)** is **True** if **x** is a **Petrol**
- **Polymorphism**
  - caller ignores which class in the supported hierarchy it is operating on

Button
- xsize
- ysize
- label_text
- interested_listeners
- xposition
- yposition
+ draw()
+ press()
+ register_callback()
+ unregister_callback()



squillero@polito.it

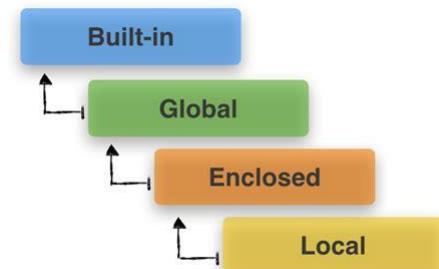
Python Quick Recap 2022

17

17

# Naming & Binding

- Names refer to objects
  - `foo = 42`  
foo is a name (not a “variable”)
- The scope defines the visibility of a name
- When a name is used, it is resolved using the nearest enclosing scope



squillero@polito.it

Python Quick Recap 2022

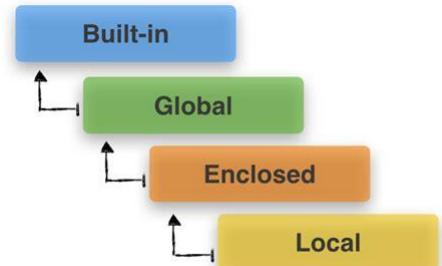
18

18

## Naming & Binding

- Names refer to objects
 

 `foo = 42`  
foo is a name (not a “variable”)
- The scope defines the visibility of a name
- When a name is used, it is resolved using the nearest enclosing scope



squillero@polito.it

Python Quick Recap 2022

19

19

## Standard Type Hierarchy

- Number
  - Integral
    - Integers (`int`)
    - Booleans (`bool`)
  - Real (`float`)
  - Complex (`complex`)
- Caveat:
  - Numbers are immutable

```

foo = 42
bar = True
baz = 4.2
qux = 4+2j
  
```

squillero@polito.it

Python Quick Recap 2022

20

20

## Standard Type Hierarchy

- Sequences
  - Immutable
    - String (**str**)
    - Tuples (**tuple**)
    - Bytes (**bytes**)
  - Mutable
    - Lists (**list**)
    - Byte Arrays (**bytearray**)

```
foo = "42"
bar = (4, 2)
baz = bytearray([0x04, 0x02])
qux = [4, 2]
tud = b'\x04\x02'
```

squillero@polito.it

Python Quick Recap 2022

21

21

## Standard Type Hierarchy

- **Mutable** Sequences vs. **Immutable** Sequences



squillero@polito.it

Python Quick Recap 2022

22

22

## Standard Type Hierarchy

- Set Types
  - Sets (**set**)
  - Frozen Sets (**frozenset**)
- Caveat:
  - Sets are **mutable**, frozen sets are **immutable**

```
foo = {4, 2}
bar = frozenset({4, 2})
```

squillero@polito.it

Python Quick Recap 2022

23

23

## Standard Type Hierarchy

- Mappings
  - Dictionaries (**dict**)

```
foo = {'Giovanni':23, 'Paola':18}
```

squillero@polito.it

Python Quick Recap 2022

24

24

## Standard Type Hierarchy

- Callable
  - Built-in functions — e.g., `len()`
  - Built-in methods — e.g., `list.append()`
  - User-defined functions — e.g., `foo()`
  - Instance methods — e.g., `foo.bar()`
  - Generator functions — e.g., anything with a `yield()`
  - Asynchronous generator functions
  - Coroutine functions

squillero@polito.it

Python Quick Recap 2022

25

25

## Standard Type Hierarchy

- None (`NoneType`)

`foo = None`

squillero@polito.it

Python Quick Recap 2022

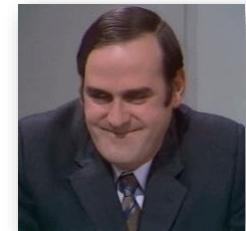
26

26

13

## Standard Type Hierarchy

- NotImplemented
- Ellipsis (...)
- Modules
- Custom classes
- Class instances
- I/O objects
- Internal types



squillero@polito.it

Python Quick Recap 2022

27

27

## Python — Quick Recap

## Basic Syntax



28

14

## Style (TL;DR)

- `module_name`
- `package_name`
- `ClassName`
- `method_name`
- `ExceptionName`
- `function_name`
- `GLOBAL_CONSTANT_NAME`
- `global_var_name`
- `instance_var_name`
- `function_parameter_name`
- `local_var_name`



<https://github.com/squillero/style>

squillero@polito.it

Python Quick Recap 2022

29

29

## Style

- Source file is UTF-8, all Unicode runes can be used
- Single underscore is a valid name (`_`)
- Safe rule:
  - Use only printable standard ASCII characters for names
  - Don't start names with single/double underscore unless you really know what you are doing
  - Append an underscore not to clash with keywords or common names, e.g., `int_` and `list_`

無 = 0
_ = 42

squillero@polito.it

Python Quick Recap 2022

30

30

# Style

- Use an automatic formatter
  - Suggested: **yapf** or **autopep8** or **black**

Python > Formatting: Provider  
Provider for formatting. Possible options include 'autopep8', 'black', and 'yapf'.  
yapf

Python > Formatting: Yapf Args  
Arguments passed in. Each argument is a separate item in the array.  
--style  
{based\_on\_style: google, column\_limit=100, blank\_line\_before\_module\_docstring=true}  
Add Item

squillero@polito.it

Python Quick Recap 2022

31

31

## Style: black vs. yapf

```
def main():
    random.seed(SEED)
    sequence = list()

    while len([
        i
        for i in range(len(sequence) - SEQUENCE_LENGTH + 1)
        if sequence[i:i + SEQUENCE_LENGTH] == sequence[-SEQUENCE_LENGTH:]
    ]) < 2:
        sequence.append(random.randint(MIN_RANDOM, MAX_RANDOM))

    print(f"Sequence length: {len(sequence)} -> Repeated pattern: {sequence[-SEQUENCE_LENGTH:]}")
```



```
def main():
    random.seed(SEED)
    sequence = list()

    while len([
        i
        for i in range(len(sequence) - SEQUENCE_LENGTH + 1)
        if sequence[i:i + SEQUENCE_LENGTH] == sequence[-SEQUENCE_LENGTH:]
    ]) < 2:
        sequence.append(random.randint(MIN_RANDOM, MAX_RANDOM))

    print(f"Sequence length: {len(sequence)} -> Repeated pattern: {sequence[-SEQUENCE_LENGTH:]}")
```



squillero@polito.it

Python Quick Recap 2022

32

32

## Comments

- Comments starts with hash (#) and ends with the line
- (Multi-line) (doc)strings might be used to comment/document the code in specific contexts

```
# This is a comment
"""

This is a multi-line docstring,
that may also help documenting the code
"""

```

squillero@polito.it

Python Quick Recap 2022

33

33

## Basic I/O: print

- Type `print()` in Visual Studio Code and wait for help

```
print()
(*values: object, sep: str | None = ..., end: str | None = ..., file: SupportsWrite[str] | None = ..., flush: bool = ...) -> None

print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
Prints the values to a stream, or to sys.stdout by default. Optional keyword arguments:
file: a file-like object (stream); defaults to the current sys.stdout.
sep: string inserted between values, default a space.
end: string appended after the last value, default a newline.
flush: whether to forcibly flush the stream.
```

squillero@polito.it

Python Quick Recap 2022

34

34

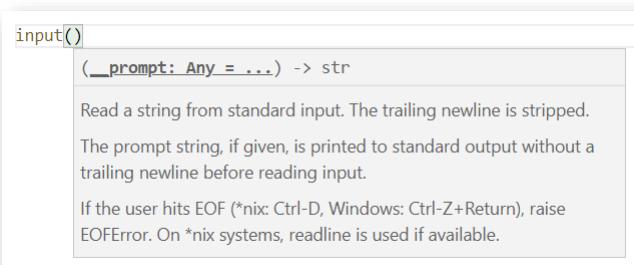
## Pythonic Approach

- Functions are simple and straightforward
- Specific “named arguments” can be optionally used to tweak the behavior

```
print("Foo", "Bar")
print("Foo", "Bar", file=sys.stderr, sep='|')
```

## Basic I/O: `input`

- Type `input()` in Visual Studio Code and wait for help



# Indentation

- Python uses indentation for defining { blocks }
- Both tabs and spaces are allowed
  - consistency is required, let alone desirable

```

1  for item in range(10):
2      print('I')
3      print('am')
4      print('a')
5      if item % 2 == 0:
6          print('funny')
7          print('and')
8          print('silly')
9      else:
10         print('dull')
11         print('and')
12         print('serious')
13     print('block')
14     print('used')
15     print('as')
16     print('example.')
17
18
19
20
21

```

squillero@polito.it

Python Quick Recap 2022

37

37

# Constructors

- Standard object constructors can be used to create empty/default objects, or to convert between types

```

foo = int()           # the default value for numbers is 0
bar = float("4.2")
baz = str(42)

```

squillero@polito.it

Python Quick Recap 2022

38

38

19

# Numbers

- Operators almost C-like:

$+$	$-$	$*$	$/$	$\%$	$//$
$+=$	$-=$	$*=$	$/=$	$%=$	$//=$

- Caveats:

$/$  always returns a floating point

$//$  always returns an integer — although not always of class `int`

Mod ( $\%$ ) always returns a result — even with a negative or float

- No self pre/post inc/dec-rement: `++` `--`

(numbers are immutable, names are not variables)

# Numeric operations

Operation	Result	Notes
<code>x + y</code>	sum of <code>x</code> and <code>y</code>	
<code>x - y</code>	difference of <code>x</code> and <code>y</code>	
<code>x * y</code>	product of <code>x</code> and <code>y</code>	
<code>x / y</code>	quotient of <code>x</code> and <code>y</code>	
<code>x // y</code>	floored quotient of <code>x</code> and <code>y</code>	(1)
<code>x % y</code>	remainder of <code>x / y</code>	(2)
<code>-x</code>	<code>x</code> negated	
<code>+x</code>	<code>x</code> unchanged	
<code>abs(x)</code>	absolute value or magnitude of <code>x</code>	
<code>int(x)</code>	<code>x</code> converted to integer	(3)(6)
<code>float(x)</code>	<code>x</code> converted to floating point	(4)(6)
<code>complex(re, im)</code>	a complex number with real part <code>re</code> , imaginary part <code>im</code> . <code>im</code> defaults to zero.	(6)
<code>c.conjugate()</code>	conjugate of the complex number <code>c</code>	
<code>divmod(x, y)</code>	the pair <code>(x // y, x % y)</code>	(2)
<code>pow(x, y)</code>	<code>x</code> to the power <code>y</code>	(5)
<code>x ** y</code>	<code>x</code> to the power <code>y</code>	(5)

## Real-number operations

Operation	Result
<code>math.trunc(x)</code>	$x$ truncated to <code>Integral</code>
<code>round(x[, n])</code>	$x$ rounded to $n$ digits, rounding half to even. If $n$ is omitted, it defaults to 0.
<code>math.floor(x)</code>	the greatest <code>Integral</code> $\leq x$
<code>math.ceil(x)</code>	the least <code>Integral</code> $\geq x$

squillero@polito.it

Python Quick Recap 2022

41

41

## Bitwise operations

Operation	Result	Notes
<code>x   y</code>	bitwise <i>or</i> of $x$ and $y$	(4)
<code>x ^ y</code>	bitwise <i>exclusive or</i> of $x$ and $y$	(4)
<code>x &amp; y</code>	bitwise <i>and</i> of $x$ and $y$	(4)
<code>x &lt;&lt; n</code>	$x$ shifted left by $n$ bits	(1)(2)
<code>x &gt;&gt; n</code>	$x$ shifted right by $n$ bits	(1)(3)
<code>~x</code>	the bits of $x$ inverted	

squillero@polito.it

Python Quick Recap 2022

42

42

# Sequences

- Ordered data structure
  - Support positional access
  - Are iterable
- Among the different sequences, the most popular are
  - Lists: mutable, heterogenous
  - Tuples: immutable, heterogenous
  - Strings: immutable, homogenous

squillero@polito.it

Python Quick Recap 2022

43

43

# Sequence operations

Operation	Result
<code>x in s</code>	True if an item of <code>s</code> is equal to <code>x</code> , else False
<code>x not in s</code>	False if an item of <code>s</code> is equal to <code>x</code> , else True
<code>s + t</code>	the concatenation of <code>s</code> and <code>t</code>
<code>s * n or n * s</code>	equivalent to adding <code>s</code> to itself <code>n</code> times
<code>s[i]</code>	<code>i</code> th item of <code>s</code> , origin 0
<code>s[i:j]</code>	slice of <code>s</code> from <code>i</code> to <code>j</code>
<code>s[i:j:k]</code>	slice of <code>s</code> from <code>i</code> to <code>j</code> with step <code>k</code>
<code>len(s)</code>	length of <code>s</code>
<code>min(s)</code>	smallest item of <code>s</code>
<code>max(s)</code>	largest item of <code>s</code>
<code>s.index(x[, i[, j]])</code>	index of the first occurrence of <code>x</code> in <code>s</code> (at or after index <code>i</code> and before index <code>j</code> )
<code>s.count(x)</code>	total number of occurrences of <code>x</code> in <code>s</code>

squillero@polito.it

Python Quick Recap 2022

44

44

## Looping over sequences

```

giovanni = "Giovanni Adolfo Pietro Pio Squillero"

for letter in giovanni:
    print(letter)

for index in range(len(giovanni)):
    print(index, giovanni[index])

for index, letter in enumerate(giovanni):
    print(index, letter)

```

- Caveat: **range()** is a class designed to efficiently generate indexes; the full constructor is:  
`class range(start, stop[, step])`

squillero@polito.it

Python Quick Recap 2022

45

45

## Looping over multiple sequences

```

for a, b in zip('GIOVANNI', 'XYZ'):
    print(f"{a}:{b}")
✓ 0.3s
G:X
I:Y
O:Z

```

Python

squillero@polito.it

Python Quick Recap 2022

46

46

## Lists and Tuples

- A list is a heterogenous, mutable sequence
- A tuple is a heterogenous, immutable sequence
- A list may contain tuples as elements, and vice versa
- Only lists may be sorted, but all iterable may be accessed through **sorted()**

```

birthday = [["Giovanni", 23, 10], ["Paola", 18, 5]]
print(birthday[0])

birthday_alt = [("Giovanni", 23, 10), ("Paola", 18, 5)]
print(birthday_alt[1][2])

birthday_alt.sort()
print(birthday_alt)

foo = (23, 10, 18, 5)
print(sorted(foo))

```

squillero@polito.it

Python Quick Recap 2022

47

47

## Sort vs. Sorted

- **sorted(foo)** returns a list containing all elements of foo sorted in ascending order
- **bar.sort()** modify bar, sorting its elements in ascending order
- Named options
  - **key**
  - **reverse** (Boolean)

squillero@polito.it

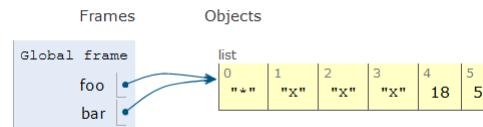
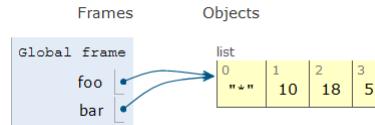
Python Quick Recap 2022

48

48

# Slices & Name binding

```
foo = [23, 10, 18, 5]
bar = foo
bar[0] = '*'
foo[1:2] = list("XXX")
```



squillero@polito.it

Python Quick Recap 2022

49

49

# Mutable-sequence operations

Operation	Result
<code>s[i] = x</code>	item <i>i</i> of <i>s</i> is replaced by <i>x</i>
<code>s[i:j] = t</code>	slice of <i>s</i> from <i>i</i> to <i>j</i> is replaced by the contents of the iterable <i>t</i>
<code>del s[i:j]</code>	same as <code>s[i:j] = []</code>
<code>s[i:j:k] = t</code>	the elements of <code>s[i:j:k]</code> are replaced by those of <i>t</i>
<code>del s[i:j:k]</code>	removes the elements of <code>s[i:j:k]</code> from the list
<code>s.append(x)</code>	appends <i>x</i> to the end of the sequence (same as <code>s[len(s):len(s)] = [x]</code> )
<code>s.clear()</code>	removes all items from <i>s</i> (same as <code>del s[:]</code> )

<code>s.copy()</code>	creates a shallow copy of <i>s</i> (same as <code>s[:]</code> )
<code>s.extend(t) or s += t</code>	extends <i>s</i> with the contents of <i>t</i> (for the most part the same as <code>s[len(s):len(s)] = t</code> )
<code>s *= n</code>	updates <i>s</i> with its contents repeated <i>n</i> times
<code>s.insert(i, x)</code>	inserts <i>x</i> into <i>s</i> at the index given by <i>i</i> (same as <code>s[i:i] = [x]</code> )
<code>s.pop() or s.pop(i)</code>	retrieves the item at <i>i</i> and also removes it from <i>s</i>
<code>s.remove(x)</code>	remove the first item from <i>s</i> where <code>s[i]</code> is equal to <i>x</i>
<code>s.reverse()</code>	reverses the items of <i>s</i> in place

squillero@polito.it

Python Quick Recap 2022

50

50

## Conditions over Sequences

- Use **any()** or **all()** to check that some/all elements in a sequence evaluates to **True**

```
print(foo)
print(any(foo))
print(all(foo))

[False, False, False, False, False, True]
True
False
```

Python

squillero@polito.it

Python Quick Recap 2022

51

51

## Strings

- Strings are immutable sequences of Unicode runes

```
string1 = "Hi!, I'm a \"string\""
string2 = 'Hi!, I\'m also a "string"'"

beatles = """John Lennon
Paul McCartney
George Harrison
Ringo Starr"""

bts = '''RM
진
슈가
제이홉
지민
뷔
정국'''
```

squillero@polito.it

Python Quick Recap 2022

52

52

# Strings

- The complete list of functions is huge
- Official documentation
  - <https://docs.python.org/3/library/stdtypes.html#textseq>
  - <https://docs.python.org/3/library/stdtypes.html#string-methods>

squillero@polito.it

Python Quick Recap 2022

53

53

# String formatting

- Several alternatives available
- Use f-strings!

```

discoverer = 'Leonhard Euler'
number = 2**31-1
year = 1772

print(f'Mersenne primes discovered by {discoverer} in {year}: {number:,}' +
      f' ({len(str(number))} digits.)')

```

✓ 0.3s

Python

Mersenne primes discovered by Leonhard Euler in 1772: 2,147,483,647 (10 digits).

- More info:

[https://docs.python.org/3/reference/lexical\\_analysis.html#f-strings](https://docs.python.org/3/reference/lexical_analysis.html#f-strings)  
<https://docs.python.org/3/library/string.html#format-spec>

squillero@polito.it

Python Quick Recap 2022

54

54

## Notable `str` methods

```
[11] "Giovanni Adolfo Pietro Pio".split()
    ✓ 0.2s
... ['Giovanni', 'Adolfo', 'Pietro', 'Pio']

[12] str.split("Giovanni Adolfo Pietro Pio")
    ✓ 0.3s
... ['Giovanni', 'Adolfo', 'Pietro', 'Pio']
```

- Caveat:  
`"bar".foo()` vs. `str.foo("bar")`

squillero@polito.it

Python Quick Recap 2022

55

55

## More notable `str` methods

```
[6] "Giovanni" + " Whoa!" * 3
    ✓ 0.4s
... 'Giovanni Whoa! Whoa! Whoa!'

▷ "|" .join(list('Giovanni'))
[8] ✓ 0.3s
... 'G|i|o|v|a|n|n|i'

[9] 'Pio' in 'Giovanni Adolfo Pietro Pio Squillero'
    ✓ 0.3s
... True
```

squillero@polito.it

Python Quick Recap 2022

56

56

## All `str` methods

```
capitalize() casefold() center(width[, fillchar])
count(sub[, start[, end]])
encode(encoding="utf-8", errors="strict")
endswith(suffix[, start[, end]]) expandtabs(tabsize=8)
find(sub[, start[, end]]) format(*args, **kwargs)
format_map(mapping) index(sub[, start[, end]]) isalnum()
isalpha() isascii() isdecimal() isdigit() isidentifier()
islower() isnumeric() isprintable() isspace() istitle()
isupper() join(iterable) ljust(width[, fillchar]) lower()
lstrip([chars]) partition(sep) removeprefix(prefix, /)
removesuffix(suffix, /) replace(old, new[, count])
rfind(sub[, start[, end]]) rindex(sub[, start[, end]])
rjust(width[, fillchar]) rpartition(sep)
rsplit(sep=None, maxsplit=-1) rstrip([chars])
split(sep=None, maxsplit=-1) splitlines([keepends])
startswith(prefix[, start[, end]]) strip([chars]) swapcase()
title() translate(table) upper() zfill(width)
```

squillero@polito.it

Python Quick Recap 2022

57

57

## Dictionary

- Heterogeneous associative array
  - Keys are required to be *hashable*, thus immutable
- Syntax similar to sequences, but no positional access

```
stone = dict()
stone['23d1364a'] = 'Mick'
stone['5465ba78'] = 'Brian'
stone['06cc49dd'] = 'Ian'
stone['c2f65729'] = 'Keith'
stone['713c0a4e'] = 'Ronnie'
stone['3ed50ef3'] = 'Charlie'

print(stone['3ed50ef3'])
```

squillero@polito.it

Python Quick Recap 2022

58

58

# Dictionary Keys and Values

```
stone.keys()
✓ 0.3s
dict_keys(['23d1364a', '5465ba78', '06cc49dd', 'c2f65729', '713c0a4e', '3ed50ef3'])

stone.values()
✓ 0.6s
dict_values(['Mick', 'Brian', 'Ian', 'Keith', 'Ronnie', 'Charlie'])

stone.items()
✓ 0.3s
dict_items([('23d1364a', 'Mick'), ('5465ba78', 'Brian'), ('06cc49dd', 'Ian'),
('c2f65729', 'Keith'), ('713c0a4e', 'Ronnie'), ('3ed50ef3', 'Charlie')])
```

squillero@polito.it

Python Quick Recap 2022

59

59

# For loops and Dictionaries

- When casted to a list or to an iterator, a dictionary is the sequence of its keys (preserving the insertion order)

```
for s in stone.keys():
    print(f"{s} -> {stone[s]}")

for s in stone:
    print(f"{s} -> {stone[s]}")

for k, v in stone.items():
    print(f"{k} -> {v}")
```

squillero@polito.it

Python Quick Recap 2022

60

60

## Sets

- Sets can be seen as dictionaries without values
- When casted to a list or to an iterator, a set is the sequence of its elements (not preserving the insertion order)
- The standard set operations can be used: **add**, **remove**, **in**, **not in**, **<= (issubset)**, **<**, **- (difference)**, **&**, **|**, **^ (symmetric\_difference)**, ...

```
foo = set("MAMMA")
bar = set("MIA")
print(foo | bar)
✓ 0.5s
```

Python

squillero@polito.it

Python Quick Recap 2022

61

61

## Copy and Delete

- **del**: delete things
  - A whole object: **del foo**
  - An element in a list: **del foo[1]**
  - An element in a dictionary: **del foo['key']**
  - An element in a set: **del foo['item']**
- **copy**: Shallow copy an object (list, dictionary, set, ...)
  - Example: **foo = bar.copy()**
  - More Pythonic alternatives may exist, e.g.: **foo = bar[:]**

squillero@polito.it

Python Quick Recap 2022

62

62

# Conditional Execution

- Generic form

  - **if [elif [... elif]]] [else]**

```
if answer == "yes" or answer == "YES":
    print("User was positive")
elif answer == "no" or answer == "NO":
    print("User was negative")
else:
    print("Dunno!?)")
```

- Caveats

  - C-Like relational operators: **== != < <= > >=**
  - Human-readable logic operators: **not and or**
  - Numeric intervals: **if 10 <= foo < 42:**
  - Special operators: **is / in**
  - Truth Value Testing: **if name:**

# While loop

- Vanilla, C-like **while**

```
foo = 117
while foo > 0:
    print(foo)
    foo //= 2
```

## Non-structured Statements

- **continue** and **break**
- **else** with **while** and **for**

```

foo = 0
bar = int(input("Start @ "))
baz = int(input("Break @ "))
while foo < 20:
    foo += 1
    if foo < bar:
        continue
    if foo == baz:
        break
    print(foo)
else:
    print("Natural end of the loop!")

```

squillero@polito.it

Python Quick Recap 2022

65

65

## pass

- The **pass** statement does nothing
- It can be used when a statement is required syntactically but the program requires no action

squillero@polito.it

Python Quick Recap 2022

66

66

## Python — Quick Recap

# Functions



67

# Functions

- Keyword **def**

```
def countdown(x):
    if not isinstance(x, int) or x <= 0:
        return False
    while x > 0:
        x /= 2
        print(x)
    return True

print(countdown("10"))
print(countdown(10))
```

- Caveat:
  - Names vs. Variables

68

## More caveats

- Functions are first-class citizen
- Function names are just “names”
- Remember scope!
- No **return** statement is equivalent to a **return None**



```
def foo():
    print("foo")

if input() == "crazy":
    def foo():
        print("crazy")

foo()
```

```
foo = input()
def bar():
    print(f"bar:{foo}")

bar()
foo = "Giovanni!"
bar()
```

squillero@polito.it

Python Quick Recap 2022

69

69

## Docstrings

- A docstring is a string literal that occurs as the first statement in a function (or module, or class, or method) definition
  - It is shown by most IDEs for helping coders
  - It becomes the **\_\_doc\_\_** special attribute of that object

```
def foo(x):
    """My FOO function"""
    pass
```

(x) -> None
My FOO function
foo()

squillero@polito.it

Python Quick Recap 2022

70

70

## Keyword and Default Arguments

- Remember scope and name binding
- Arguments may be optional if a default is provided
- Keyword arguments can be in any order

```

foo = 1
bar = 2
baz = 3

def silly_function(bar, baz=99):
    print(foo, bar, baz)

silly_function(23)
silly_function(23, baz=10)
silly_function(baz=10, bar=23)

```

squillero@polito.it

Python Quick Recap 2022

71

71

## Positional vs. Keyword Arguments

- Arguments preceding the ‘/’ are positional-only and cannot be specified to using keywords
- Arguments following the ‘\*’ must be specified using keywords
- Arguments between ‘/’ and ‘\*’ might be either positional or keyword

```

def foo(x, y, /, foo=1, bar=2, *, baz=3):
    print(f"POSITIONAL: x={x}; y={y}")
    print(f"foo={foo}; bar={bar}")
    print(f"KEYWORD ONLY: baz={baz}")

```

Python

✓ 0.3s

squillero@polito.it

Python Quick Recap 2022

72

72

## \*args and \*\*kwargs

- When a formal parameter is in the form **\*name** it receives a tuple with all the remaining positional arguments

```
def foo(x, y, *args):
    print(f"POSITIONAL: x={x}; y={y}")
    print(f"{type(args)} = {args}")

foo(23, 10, 18, 5)
✓ 0.2s
```

Python

```
POSITIONAL: x=23; y=10
<class 'tuple'> = (18, 5)
```

squillero@polito.it

Python Quick Recap 2022

73

73

## \*args and \*\*kwargs

- When a formal parameter is in the form **\*\*name** it receives a dictionary with all the remaining keyword arguments
- The argument **\*\*name** must follow **\*name**

```
def foo(x, y, *args, foo, **kwargs):
    print(f"args: {type(args)} = {args}")
    print(f"kwargs: {type(kwargs)} = {kwargs}")

foo(23, 10, 18, 5, foo='foo', bar='bar', baz='baz')
✓ 0.2s
```

Python

```
args: <class 'tuple'> = (18, 5)
kwargs: <class 'dict'> = {'bar': 'bar', 'baz': 'baz'}
```

squillero@polito.it

Python Quick Recap 2022

74

74

## True Pythonic Scripts

- Define all global ‘constants’ first, then functions
- Test \_\_name\_\_

```
GLOBAL_CONSTANT = 42

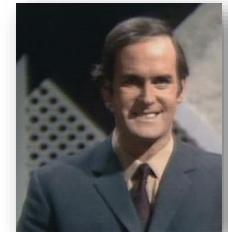
def foo():
    pass

def main():
    print(foo)
    pass

if __name__ == '__main__':
    # parse command line
    # setup logging
    main()
```

squillero@polito.it

Python Quick Recap 2022



75

75

## Callable Objects

- Names can refer to functions (*callable* objects)
- Functions are first class citizen

```
def foo(bar):
    print(f"foo{bar}")

qux = foo
qux('123')
✓ 0.3s
```

Python

squillero@polito.it

Python Quick Recap 2022

76

76

## Lambda Keyword

- Lambda expressions are short (1 line), usually simple, and anonymous functions. They can be used to calculate values

```
foo = lambda x: 2**x
foo(10)
✓ 0.3s
1024
```

Python

- or perform simple tasks

```
foo = lambda x: print(f"foo{str(x)}")
foo(10)
✓ 0.3s
foo10
```

Python

squillero@polito.it

Python Quick Recap 2022

77

77

## Lambda Keyword

- Lambda expressions are quite useful to define simple, scratch functions to be used as argument in behavioral parametrization, e.g., as **key** for the **sort()** function

```
sorted_keys = sorted(my_dict, key=lambda k: my_dict[k])
```

squillero@polito.it

Python Quick Recap 2022

78

78

39

# Closures and Scope

- Consider how names are resolved

```
foo = 42
func = lambda x: foo + x
print(func(10))
foo = 1_000
print(func(10))
✓ 0.3s
```

Python

52  
1010

squillero@polito.it

Python Quick Recap 2022



79

79

# Closures and Scope

- Consider how names are resolved

```
def make_inc(number):
    return lambda x: number + x

i10 = make_inc(10)
i500 = make_inc(500)

print(i10(1), i500(1))

✓ 0.4s
```

Python

11 501

squillero@polito.it

Python Quick Recap 2022



80

80

## Python — Quick Recap

# List Comprehensions & Generators



81

## List Comprehensions

- A concise way to create lists

```
[x for x in range(10)]
✓ 0.4s
```

Python

```
[x for x in range(50) if x % 3 == 0]
✓ 0.3s
```

Python

```
[(x, x**y) for x in range(2, 4) for y in range(4, 7)]
✓ 0.4s
```

Python

## Generators

- Like list comprehension, but elements are not actually calculated unless explicitly required by **next()**

squillero@polito.it

```
foo = (x**x for x in range(100_000))
foo
✓ 0.3s
<generator object <genexpr> at 0x0000026D42A1C0B0>
Python
```

```
for x in range(3):
|   print(f"next(foo):{next(foo)}")
✓ 0.3s
1
1
4
Python
```

```
for x in range(3):
|   print(f"next(foo):{next(foo)}")
✓ 0.4s
27
256
Python
```

Python Quick Recap 2022

83

83

## Generators

- Can be quite effective inside **any()** or **all()**

squillero@polito.it

Python Quick Recap 2022

84

```
any([n % 23 == 0 for n in range(1, 1_000_000_000)])
✓ 63.8s
True
Python
```

```
any(n % 23 == 0 for n in range(1, 100_000_000))
✓ 0.3s
True
Python
```

84

## Generators

- Can be used to create lists and sets

```
numbers = set(a*b for a in range(11) for b in range(11))
print("All integral numbers that can be expressed as a*b with a and b less than 10: " +
      ' '.join(str(_) for _ in sorted(numbers)))
✓ 0.3s
```

Python

```
All integral numbers that can be expressed as a*b with a and b less than 10: 0 1 2 3 4 5
6 7 8 9 10 12 14 15 16 18 20 21 24 25 27 28 30 32 35 36 40 42 45 48 49 50 54 56 60 63 64
70 72 80 81 90 100
```

squillero@polito.it

Python Quick Recap 2022

85

85

## Python — Quick Recap

# Exceptions



86

# Exceptions

- Like (almost) all modern languages, Python implements a mechanism for handling unexpected events in a smooth way through “exceptions”

```
try:
    val = risky_code()
except ValueError:
    val = None
except Exception as problem:
    logging.critical(f"Yeuch: {problem}")

if val == None:
    raise ValueError("Yeuch, invalid value")
```

# Notable Exceptions

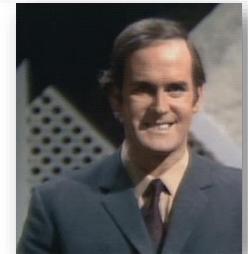
- Exception**
- ArithmetError**
  - **OverflowError**, **ZeroDivisionError**, **FloatingPointError**
- LookupError**
  - **IndexError**, **KeyError**
- OSError**
  - System-related error, including I/O failures
- UnicodeEncodeError**, **UnicodeDecodeError** and **UnicodeTranslateError**
- ValueError**

## Assert

- Check specific conditions at run-time
- Ignored if compiler is optimizing (`-O` or `-OO`)
- Generate an **AssertionError**

```
assert val != None, "Yeuch, invalid val"
```

- Advice: Use a lot of asserts in your code



squillero@polito.it

Python Quick Recap 2022

89

89

## Python — Quick Recap

# Modules



90

# Modules

- Python code libraries are organized in modules
- Names in modules can be imported in several way

```
import math
print(math.sqrt(2))
✓ 0.4s
1.4142135623730951
```

Python

```
from math import sqrt
from cmath import sqrt as c_sqrt
print(sqrt(2), c_sqrt(-2))
✓ 0.4s
1.4142135623730951 1.4142135623730951j
```

Python

squillero@polito.it

Python Quick Recap 2022

91

91

# Notable Modules

```
import logging

logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s %(levelname)s: %(message)s',
    datefmt=['%H:%M:%S'],
)

logging.debug("For debugging purpose")
logging.info("Verbose information")
logging.warning("Watch out, it looks important")
logging.error("We have a problem")
logging.critical("Armageddon was yesterday, now we have a real problem...")
✓ 0.8s
```

Python

squillero@polito.it

Python Quick Recap 2022

92

92

## Notable Modules

- Warnings that alert the user of some important condition that doesn't warrant raising an exception and terminating the program (e.g., uses of a obsolete feature)
- Caveat: **logging** vs. **warnings**

squillero@polito.it

Python Quick Recap 2022

93

93

## Notable Modules

- Mathematical functions, use **cmath** for complex numbers

squillero@polito.it

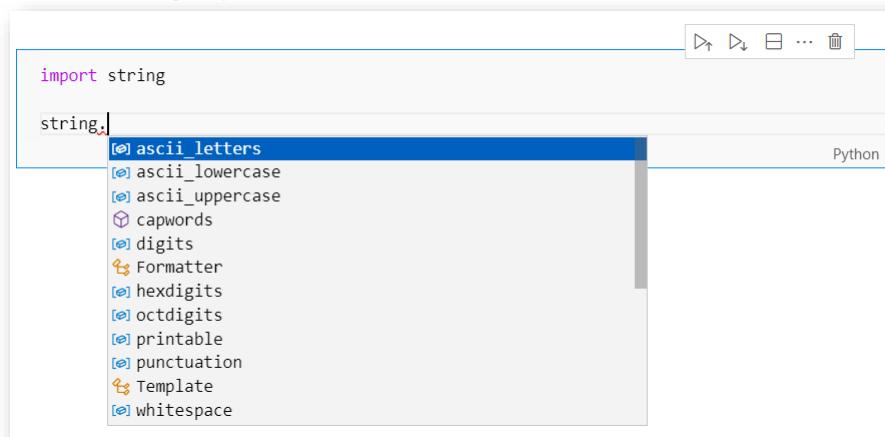
Python Quick Recap 2022

94

94

## Notable Modules

- Common string operations and constants



The screenshot shows a code editor window with the following code:

```
import string

string.
```

A tooltip-like pop-up window lists the members of the `string` module's `__all__` list:

- [e] ascii\_letters
- [e] ascii\_lowercase
- [e] ascii\_uppercase
- [i] capwords
- [e] digits
- [e] Formatter
- [e] hexdigits
- [e] octdigits
- [e] printable
- [e] punctuation
- [e] Template
- [e] whitespace

The word `ascii_letters` is highlighted with a blue selection bar.

squillero@polito.it

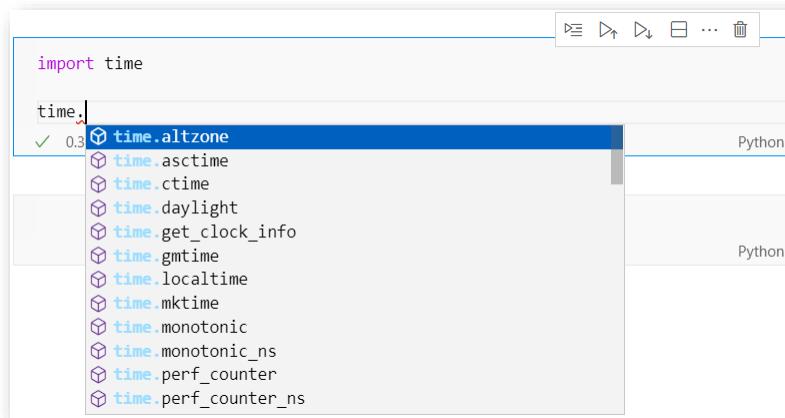
Python Quick Recap 2022

95

95

## Notable Modules

- Various time-related functions



The screenshot shows a code editor window with the following code:

```
import time

time.
```

A tooltip-like pop-up window lists the members of the `time` module's `__all__` list:

- ✓ 0.3 [e] time.altzone
- [i] time.asctime
- [i] time.ctime
- [i] time.daylight
- [i] time.get\_clock\_info
- [i] time.gmtime
- [i] time.localtime
- [i] time.mktime
- [i] time.monotonic
- [i] time.monotonic\_ns
- [i] time.perf\_counter
- [i] time.perf\_counter\_ns

The word `time.altzone` is highlighted with a blue selection bar.

squillero@polito.it

Python Quick Recap 2022

96

96

## Notable Modules: **time**

- **perf\_counter** / **perf\_counter\_ns**
  - Clock with the highest available resolution to measure a short duration, it does include time elapsed during sleep and is system-wide
  - Only the difference between the results of two calls is valid
- **process\_time** / **process\_time\_ns**
  - Sum of the system and user CPU time of the current process. It does not include time elapsed during sleep
  - Only the difference between the results of two calls is valid

squillero@polito.it

Python Quick Recap 2022

97

97

## Notable Modules: **time**

- Measure performances:
  - Use **process\_time** in a script
  - Use **%timeit** in a notebook

```
%timeit fibonacci_numbers = [n for n, _ in zip(fibonacci(), range(100))]
```

✓ 7.9s

Python

9.76 µs ± 8.35 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)

squillero@polito.it

Python Quick Recap 2022

98

98

## Notable Modules: **time**

- **sleep**

- Suspend execution of the calling thread for the given number of seconds.

squillero@polito.it

Python Quick Recap 2022

99

99

## Notable Modules : **pprint**

- Data pretty printer, sometimes a good replacement for **print**
  - Notez bien: tons of customizations importing the whole module

```
from pprint import pprint

numbers = [set(y+x**y for y in range(x)) for x in range(1, 10)]
pprint(numbers)
✓ 0.4s
```

Python

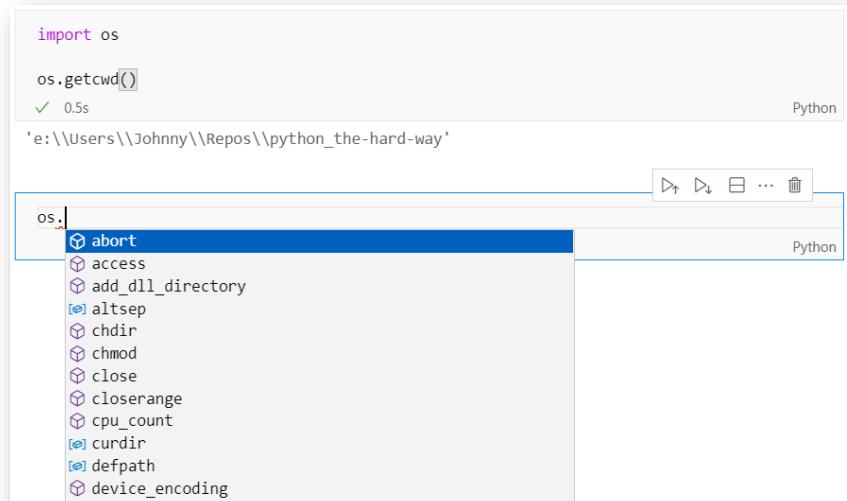
squillero@polito.it

100

100

## Notable Modules: **os**

- Miscellaneous operating system interfaces



```
import os
os.getcwd()
✓ 0.5s
'e:\\\\Users\\\\Johnny\\\\Repos\\\\python_the-hard-way'
```

The screenshot shows a Python code editor with the following code:

```
import os
os.getcwd()
✓ 0.5s
'e:\\\\Users\\\\Johnny\\\\Repos\\\\python_the-hard-way'
```

A tooltip or dropdown menu is open over the `os.` part of the code, listing various methods of the `os` module. The method `abort()` is highlighted in blue, indicating it is currently selected or being used.

squillero@polito.it

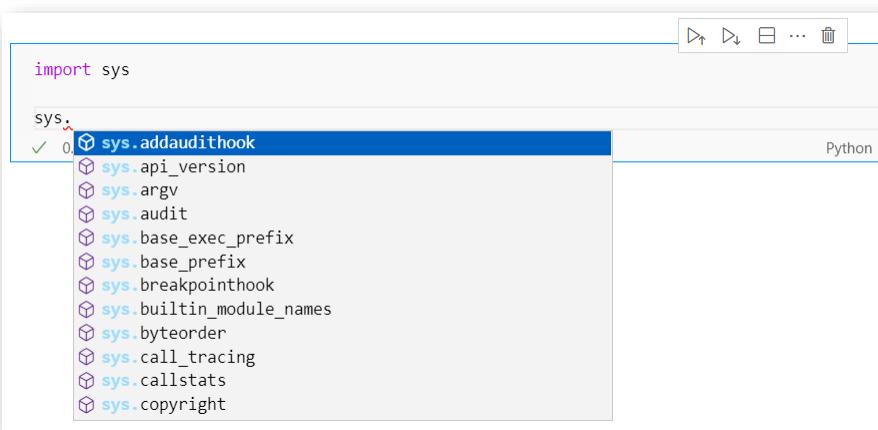
Python Quick Recap 2022

101

101

## Notable Modules: **sys**

- System-specific parameters and functions



```
import sys
sys.
✓ 0.5s
sys.addaudithook
sys.api_version
sys.argv
sys.audit
sys.base_exec_prefix
sys.base_prefix
sys.breakpointhook
sys.builtin_module_names
sys.byteorder
sys.call_tracing
sys.callstats
sys.copyright
```

The screenshot shows a Python code editor with the following code:

```
import sys
sys.
✓ 0.5s
sys.addaudithook
sys.api_version
sys.argv
sys.audit
sys.base_exec_prefix
sys.base_prefix
sys.breakpointhook
sys.builtin_module_names
sys.byteorder
sys.call_tracing
sys.callstats
sys.copyright
```

A tooltip or dropdown menu is open over the `sys.` part of the code, listing various methods of the `sys` module. The method `addaudithook()` is highlighted in blue, indicating it is currently selected or being used.

squillero@polito.it

Python Quick Recap 2022

102

102

## Notable Modules: **re**

- Regular expression operations

```
import re
^
re.
✓ re.A
<module 're' (built-in)>
  ▾ re.ASCII
  ▾ re.compile
  ▾ re.copyreg
  ▾ re.DEBUG
  ▾ re.DOTALL
  ▾ re.enum
  ▾ re.error
  ▾ re.escape
  ▾ re.findall
  ▾ re.finditer
  ▾ re.fullmatch
```

The screenshot shows a Python code editor interface. On the left, there is a code editor window containing the following code:

```
import re
^
re.
✓ re.A
<module 're' (built-in)>
  ▾ re.ASCII
  ▾ re.compile
  ▾ re.copyreg
  ▾ re.DEBUG
  ▾ re.DOTALL
  ▾ re.enum
  ▾ re.error
  ▾ re.escape
  ▾ re.findall
  ▾ re.finditer
  ▾ re.fullmatch
```

The word 're.A' is highlighted in blue, indicating it is selected or being typed. To the right of the code editor are two empty windows labeled 'Python'.

squillero@polito.it

Python Quick Recap 2022

103

103

## Notable Modules: **itertools**

- Real Python programmers do not love double loops
- Use **itertools** for efficient looping

Examples	Results
<code>product('ABCD', repeat=2)</code>	AA AB AC AD BA BB BC BD CA CB CC CD DA DB DC DD
<code>permutations('ABCD', 2)</code>	AB AC AD BA BC BD CA CB CD DA DB DC
<code>combinations('ABCD', 2)</code>	AB AC AD BC BD CD
<code>combinations_with_replacement('ABCD', 2)</code>	AA AB AC AD BB BC BD CC CD DD

squillero@polito.it

Python Quick Recap 2022

104

104

## More `itertools`

- Infinite loops
  - `count`, `cycle`, `repeat`
- Terminating on the shortest sequence
  - `accumulate`, `chain`, `chain.from_iterable`, `compress`, `dropwhile`, `filterfalse`, `groupby`, `islice`, `starmap`, `takewhile`, `tee`, `zip_longest`

squillero@polito.it

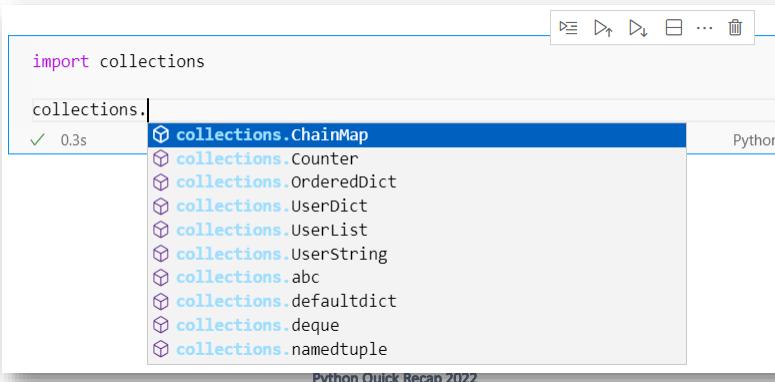
Python Quick Recap 2022

105

105

## Notable Modules: `collection`

- The module `collection` contains specialized container datatypes providing alternatives to Python's general purpose built-in containers



```
import collections

collections.
```

0.3s

- `collections.ChainMap`
- `collections.Counter`
- `collections.OrderedDict`
- `collections.UserDict`
- `collections.UserList`
- `collections.UserString`
- `collections.abc`
- `collections.defaultdict`
- `collections.deque`
- `collections.namedtuple`

squillero@polito.it

Python Quick Recap 2022

106

106

## Notable Modules: **collection**

<code>namedtuple()</code>	factory function for creating tuple subclasses with named fields
<code>deque</code>	list-like container with fast appends and pops on either end
<code>ChainMap</code>	dict-like class for creating a single view of multiple mappings
<code>Counter</code>	dict subclass for counting hashable objects
<code>OrderedDict</code>	dict subclass that remembers the order entries were added
<code>defaultdict</code>	dict subclass that calls a factory function to supply missing values
<code>UserDict</code>	wrapper around dictionary objects for easier dict subclassing
<code>UserList</code>	wrapper around list objects for easier list subclassing
<code>UserString</code>	wrapper around string objects for easier string subclassing

squillero@polito.it

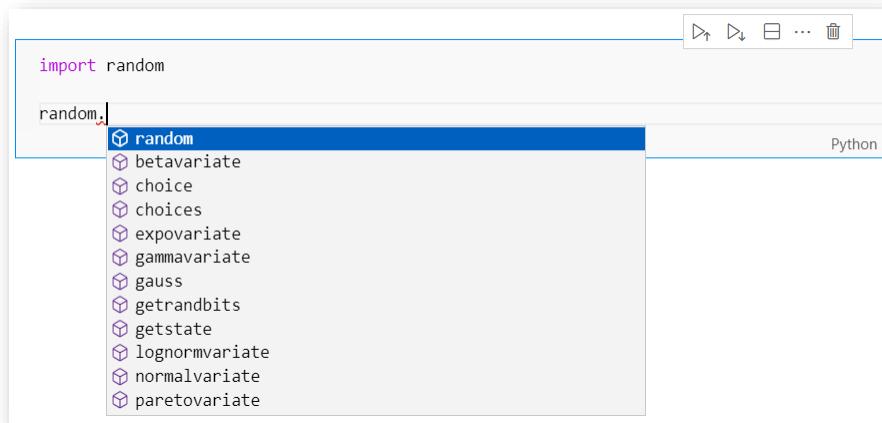
Python Quick Recap 2022

107

107

## Notable Modules: **random**

- Generate pseudo-random numbers



The screenshot shows a code editor interface with a Python file open. The code at the top is:

```
import random
```

Below the code, the cursor is positioned after the word "random" in the import statement. A tooltip or documentation sidebar is displayed, showing the following list of methods:

- random.
- random.betavariate
- random.choice
- random.choices
- random.expovariate
- random.gammavariate
- random.gauss
- random.getrandbits
- random.getstate
- random.lognormvariate
- random.normalvariate
- random.paretovariate

The sidebar has a "Python" tab at the bottom right.

squillero@polito.it

Python Quick Recap 2022

108

108

## Notable Modules: argparse

- Parse command-line arguments

The screenshot shows a code editor interface with a Python file open. The code at the top is:

```
import argparse
```

Below it, the user has typed "argparse." followed by a period, triggering a completion dropdown. The dropdown is titled "argparse.Action" and lists several members of the argparse module, including ArgumentDefaultsHelpFormatter, ArgumentError, ArgumentParser, ArgumentTypeError, FileType, HelpFormatter, MetavarTypeHelpFormatter, Namespace, ONE\_OR\_MORE, OPTIONAL, and PARSER. The "argparse.Action" entry is highlighted with a blue bar.

squillero@polito.it

Python Quick Recap 2022

109

109

## Notable Modules: argparse

- Quite complex and powerful
- Help and recipes available from [python.org](https://python.org)

```
parser = argparse.ArgumentParser()
parser.add_argument('-v', '--verbose', action='count', default=0, help='increase log verbosity')
parser.add_argument('-d',
                    '--debug',
                    action='store_const',
                    dest='verbose',
                    const=2,
                    help='log debug messages (same as -vv)')
args = parser.parse_args()

if args.verbose == 0:
    logging.getLogger().setLevel(logging.WARNING)
elif args.verbose == 1:
    logging.getLogger().setLevel(logging.INFO)
elif args.verbose == 2:
    logging.getLogger().setLevel(logging.DEBUG)
```

squillero@polito.it

Python Quick Recap 2022

110

110

## Python — Quick Recap

i/o



111

## Working with files

- As simple as

```
try:
    with open('file_name', 'r') as data_input:
        # read from data_input
        pass
except OSError as problem:
    logging.error(f"Can't read: {problem}")
```

- Caveat:
  - Use **try/except** to handle errors,  
**with** to dispose resource automagically
  - Default encoding is '**utf-8**'

## Open mode

- The mode may be specified setting the named parameter **mode** to 1 or more characters

Character	Meaning
'r'	open for reading (default)
'w'	open for writing, truncating the file first
'x'	open for exclusive creation, failing if the file already exists
'a'	open for writing, appending to the end of the file if it exists
'b'	binary mode
't'	text mode (default)
'+'	open for updating (reading and writing)

## Under the hood

- If mode is not *binary*, a **TextIOWrapper** is used
  - buffered text stream providing higher-level access to a **BufferedIOBase** buffered binary stream
- If mode is *binary* and *read*, a **BufferedReader** is used
  - buffered binary stream providing higher-level access to a readable, non seekable **RawIOBase** raw binary stream
- If mode is *binary* and *write*, a **BufferedWriter** is used
  - buffered binary stream providing higher-level access to a writeable, non seekable **RawIOBase** raw binary stream

# Reading/Writing text files

- **read(size)**
  - Reads up to n bytes, default slurp the whole file
- **readline()**
  - Reads 1 line
- **readlines()**
  - Reads the whole file and returns a list of lines
- **write(text)**
  - Write text into the file, no automatic newline is added
- **tell()/seek(offset)**
  - Gets/set the position inside a file



squillero@polito.it

Python Quick Recap 2022

115

## Example

```

try:
    with open('input.txt') as input, open('output.txt', 'w') as output:
        for line in input:
            output.write(line)
except OSError as problem:
    logging.error(problem)

```

squillero@polito.it

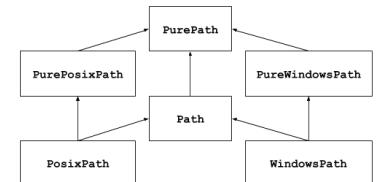
Python Quick Recap 2022

116

116

# Paths

- To manipulate paths in a somewhat portable way across different operating systems, use either
  - `os.path`
  - `pathlib` (more object oriented)
- Standard function names, such as `basename()` or `isfile()`
- Paths are always *local* path, to force:
  - `posixpath` (un\*x)
  - `ntpath` (windoze)



squillero@polito.it

Python Quick Recap 2022

117

117

# Module: `pickle`

- Binary serialization and de-serialization of Python objects

```

import pickle

foo = get_really_complex_data_structure()
pickle.dump(foo, open('dump.p', 'wb'))
bar = pickle.load(open('dump.p', 'rb'))
  
```

- Caveats:

- File operations should be inside `try/catch`
- Use `protocol=0` to get a human-readable pickle file
- The module is not **secure!** An attacker may tamper the pickle file and make `unpickle` execute arbitrary code



squillero@polito.it

Python Quick Recap 2022

118

118

## Read CSV

- As standard file

119

119

## Notable Modules: `csv`

- With the CSV module (*sniffing* the correct format)

```
import csv
file = os.path.join(os.getcwd(), 'data_files', 'big_graphs_benchmark.csv')
try:
    with open(file) as csv_file:
        dialect = csv.Sniffer().sniff(csv_file.read())
        csv_file.seek(0)
        for x in csv.reader(csv_file, dialect=dialect):
            print(x)
except OSError as problem:
    logging.error(f"Can't read {file.name}: {problem}")
✓ 0.8s
```

120

120

## Notable Modules: `configparser`

- Handle (read and write) standard config files

```
import configparser

config = configparser.ConfigParser()
config.read(os.path.join(os.getcwd(), 'data_files', 'sample-config.ini'))
print(config['Simple Values']['key'])
print('no key' in config['Simple Values'])
print('spaces in values' in config['Simple Values'])
```

Python

value  
False  
True

1 [Simple Values]  
2 key=value  
3 spaces in keys=allowed  
4 spaces in values=allowed as well  
5 spaces around the delimiter = obviously  
6 you can also use : to delimit keys from values

squillero@polito.it

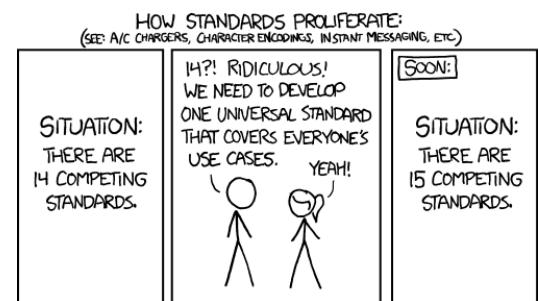
Python Quick Recap 2022

121

121

## Alternatives to `configparser`?

- toml**
  - Library for Tom's Obvious, Minimal Language
- json**
  - JSON encoder and decoder



squillero@polito.it

Python Quick Recap 2022

122

122

## Notable Modules: `toml`

```
pyproject = toml.load('pyproject.toml')

pprint(pyproject)

{'build-system': {'build-backend': 'poetry.core.masonry.api',
                  'requires': ['poetry-core>=1.0.0']},
 'tool': {'black': {'include': '\\.ipynb', 'line-length': 120},
          'poetry': {'authors': ['Giovanni Squillero <squillero@polito.it>'],
                     'dependencies': {'black': {'extras': ['jupyter'],
                                              'version': '>=22.6'},
                                      'jupyter': '>=1.0.0',
                                      'numpy': '>=1.23',
                                      'pandas': '>=1.4',
                                      'python': '>=3.10',
                                      'scikit-learn': '^1.1.2',
                                      'seaborn': '^0.11.2'}}}}
```

squillero@polito.it

Python Quick Recap 2022



```
[tool.yapf]
based_on_style = "google"
blank_line_before_module_docstring = true
column_limit = 120

[tool.yapfignore]
ignore_patterns = [
    ".*/**",
]

[tool.black]
include = "\\.\.ipynb"
line-length = 120

[tool.poetry]
authors = ["Giovanni Squillero <squillero@polito.it>"]
description = "Common environment for Squillero's 10k foot views"
license = "Proprietary"
name = "10k"
version = "0.1.0"

[tool.poetry.dependencies]
black = {extras = ["jupyter"], version = ">=22.6"}
jupyter = ">=1.0.0"
numpy = ">=1.23"
pandas = ">=1.4"
python = ">=3.10"
scikit-learn = "^1.1.2"
```

123

## Notable Modules: `json`

```
print(json.dumps(toml.load('pyproject.toml'), sort_keys=True, indent=4))

{
    "build-system": {
        "build-backend": "poetry.core.masonry.api",
        "requires": [
            "poetry-core>=1.0.0"
        ]
    },
    "tool": {
        "black": {
            "include": "\\.\.ipynb",
            "line-length": 120
        },
        "poetry": {
            "authors": [
                "Giovanni Squillero <squillero@polito.it>"
            ],
            "dependencies": {
                "black": {
                    "extras": [
                        "jupyter"
                    ]
                }
            }
        }
    }
}
```

squillero@polito.it

Python Quick Recap 2022

Add Code Cell | Add Markdown Cell

```
original_dict = toml.load('pyproject.toml')
json_string = json.dumps(original_dict)
json_dict = json.loads(json_string)
original_dict == json_dict
```

True

124

124

## Python — Quick Recap

# Linters



125

## Linting?

**lint** [from Unix's `lint(1)`, named for the bits of fluff it supposedly picks from programs] 1. /vt./ To examine a program closely for style, language usage, and portability problems, esp. if in C, esp. if via use of automated analysis tools, most esp. if the Unix utility `lint(1)` is used. This term used to be restricted to use of `lint(1)` itself, but (judging by references on Usenet) it has become a shorthand for desk check at some non-Unix shops, even in languages other than C. Also as /v./ `delint`. 2. /n./ Excess verbiage in a document, as in “This draft has too much lint”.

# pylint

- A static code-analysis tool which looks for programming errors, helps enforcing a coding standard, sniffs for code smells and offers simple refactoring suggestions

```
pip install -U pylint
```

```
(base) λ pylint ex07_random.py
*****
Module ex07_random
ex07_random.py:1:0: C0114: Missing module docstring (missing-module-docstring)
ex07_random.py:14:0: C0116: Missing function or method docstring (missing-function-docstring)

-----
Your code has been rated at 8.75/10 (previous run: 9.38/10, -0.62)
```

squillero@polito.it

127

# flake8

- A tool for enforcing style consistency across

```
pip install -U flake8
```

```
(base) λ flake8 ex07_random.py
ex07_random.py:24:80: E501 line too long (99 > 79 characters)
```

```
(base) λ flake8 --max-line-length=100 ex07_random.py
```

squillero@polito.it

Python Quick Recap 2022

128

# bandit

- A static code-analysis tool which finds common security issues in Python code

```
pip install -U bandit
```

squillero@polito.it

Python Quick Recap 2022

129

```
(base) A bandit my/random.py
[main] INFO profile include tests: None
[main] INFO profile exclude tests: None
[main] INFO cli include tests: None
[main] INFO cli exclude tests: None
[main] INFO running on Python 3.8.11
[main] INFO unable to find qualified name for module: ext_random.py
Run started 2021-09-05 13:48:40.508267

Test results:
+> issue: [B311:blacklist] Standard pseudo-random generators are not suitable for security/cryptographic use. Confidence: High
  location: ext_random.py:22
  More info: https://bandit.readthedocs.io/en/latest/blacklists/blacklist_calls.html#B311-random
    21      }) < 2:
    22          sequence.append(random.randint(MIN_RANDOM, MAX_RANDOM))
    23

Code scanned:
  Total lines of code: 19
  Total lines skipped (anosec): 0

Run metrics:
  Total issues (by severity):
    Undefined: 0.0
    Low: 1.0
    Medium: 0.0
    High: 0.0
  Total issues (by confidence):
    Undefined: 0.0
    Low: 0.0
    Medium: 0.0
    High: 1.0
  Files skipped (0):
```

129

# mypy

- Mypy is an “optional” static type checker for Python that aims to combine the benefits of dynamic (i.e., *duck*) typing and static typing.

```
pip install -U mypy
```

squillero@polito.it

```
> $ mypy .
base.py:70: error: Missing return statement
base.py:76: error: Missing return statement
base.py:82: error: Missing return statement
base.py:93: error: Argument 1 of "__eq__" is incompatible with supertype "object"; supertype defines the argument type as "object"
base.py:93: note: This violates the Liskov substitution principle
base.py:93: note: See https://mypy.readthedocs.io/en/stable/common_issues.html#incompatible-overrides
base.py:93: note: It is recommended for "__eq__" to work with arbitrary objects, for example:
base.py:93: note:     def __eq__(self, other: object) -> bool:
base.py:93: note:         if not isinstance(other, Base):
base.py:93: note:             return NotImplemented
base.py:93: note:         return <logic to compare two Base instances>
base.py:96: error: Argument 1 of "__ne__" is incompatible with supertype "object"; supertype defines the argument type as "object"
base.py:96: note: This violates the Liskov substitution principle
base.py:96: note: See https://mypy.readthedocs.io/en/stable/common_issues.html#incompatible-overrides
simple.py:40: error: Argument 1 of "is_equal" is incompatible with supertype "Base"; supertype defines the argument type as "Base"
```

130