

Python

THE HARD WAY



1

*** DRAFT v0.1 ***

https://github.com/squillero/python_the-hard-way/

Copyright © 2021 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.

2

Why Python?

- High-level programming language, truly portable
- Actively developed, open-source and community-driven
- Batteries included, huge code base
- Steep learning curve, easy to learn and to use
- Powerful as a scripting language
- Support both programming in the large and in the small
- Can be used interactively
- De-facto standard in some domain (e.g., data science)

squillero@polito.it

Python — The Hard Way

3

3

Why “the Hard Way”?

- No-nonsense Python for programmers
- Not a *gentle introduction*
- Not the usual *Dummy’s guide to...*



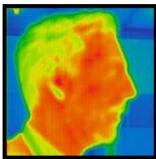
squillero@polito.it

Python — The Hard Way

4

4

Who is this Giovanni Squillero, anyway?



Associate Professor
of Computer Science
@ PoliTO (DAUIN)

- Courses:
 - Computer Sciences
 - Computational Intelligence
 - Futuro del Lavoro
 - Mimetic Learning
 - Computer Architectures
[in Uzbekistan]

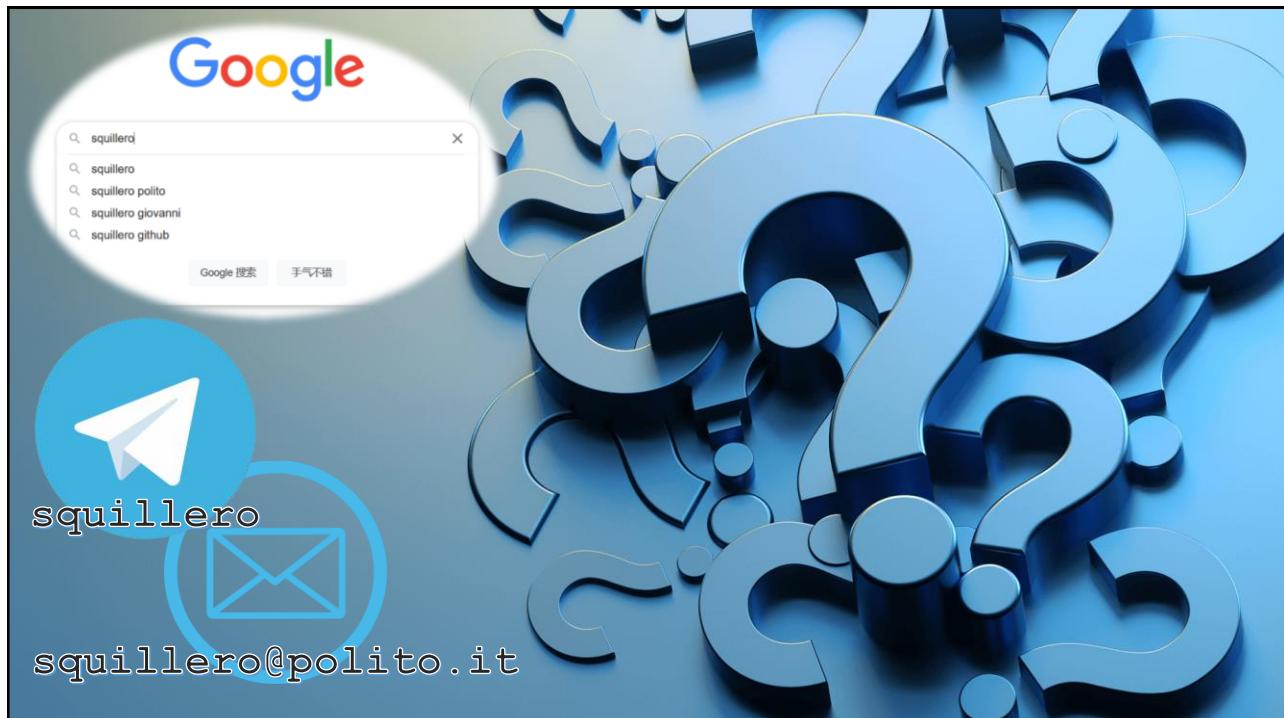
squillero@polito.it

- Current ML projects
 - Test & Validation
 - Analysis of Mass Spectra (COVID)
- Current CI projects
 - Antimicrobial susceptibility from DNA
 - *ARTificial Intelligence* (AI4MUSE)
- Research (not applied)
 - Diversity promotion in EC
 - Feature-selection is ML
 - EDA
 - Games/Economy
 - Multi-agent systems

[Python — The Hard Way](#)

5

5



6

3

Python — The Hard Way

Set-up



7

Material

- Official online documentation
 - <https://docs.python.org/3/>
 - <https://www.python.org/dev/peps/>
- Spare online resources
 - <https://stackoverflow.com/questions/tagged/python>
 - <https://www.google.com/search?q=python>
 - <https://pythontutor.com>
- Course repo
 - https://github.com/squillero/python_the-hard-way

Getting Python

- Official Python downloads
 - <https://www.python.org/downloads/>
- On Linux
 - Get the default package from the distro
 - Then use **Virtualenv** (<https://virtualenv.pypa.io/>)
- On Windows and MacOS
 - Install **Anaconda** (<https://www.anaconda.com/products/individual>)
 - If size is a **real** issue, also consider **micromamba**
- If everything else fail:
 - Try **ActivePython** (<https://www.activestate.com/products/python/>)

squillero@polito.it

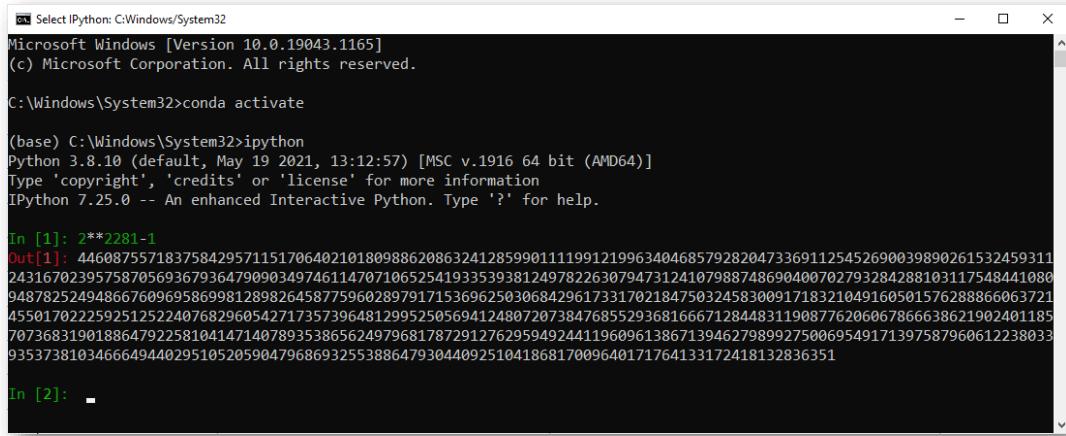
Python — The Hard Way

9

9

Getting Python

- Let's test Python calculating the 17th Mersenne prime



The screenshot shows a Windows command prompt window titled "Select IPython: C:\Windows\System32". The window displays the following Python session:

```

Select IPython: C:\Windows\System32
Microsoft Windows [Version 10.0.19043.1165]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32>conda activate
(base) C:\Windows\System32>ipython
Python 3.8.10 (default, May 19 2021, 13:12:57) [MSC v.1916 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 7.25.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: 2**2281-1
Out[1]: 4460875571837584295711517064021018098862086324128599011119912199634046857928204733691125452690039890261532459311
243167023957587056936793647909034974611470710652541933539381249782263079473124107988748690400702793284288103117548441080
948782524948667609695869981289826458775960289791715369625830684296173317021847503245830091718321849160501576288866063721
455017022259251252240768296054271735739648129952505694124807207384768552936816667128448311908776206067866638621902401185
70736831901886479225810414714078935386562497968178729127629594924411960961386713946279899275006954917139758796061223803
9353738103466649440295105205904796869325538864793044092510418681700964017176413317241813283631

In [2]: -

```

squillero@polito.it

Python — The Hard Way

10

10

IDE

- Install Visual Studio Code and the Python Extension



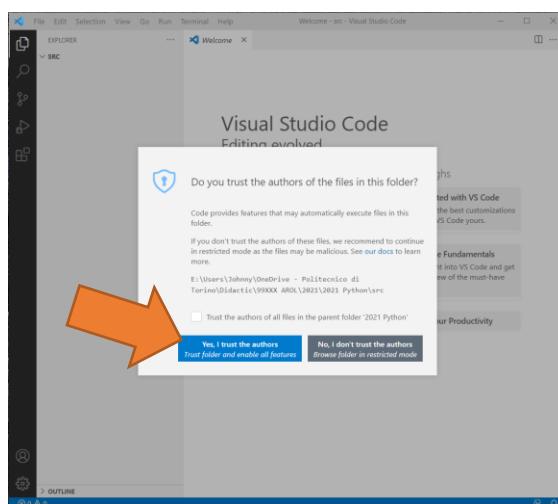
squillero@polito.it

Python — The Hard Way

11

11

Open a “directory”



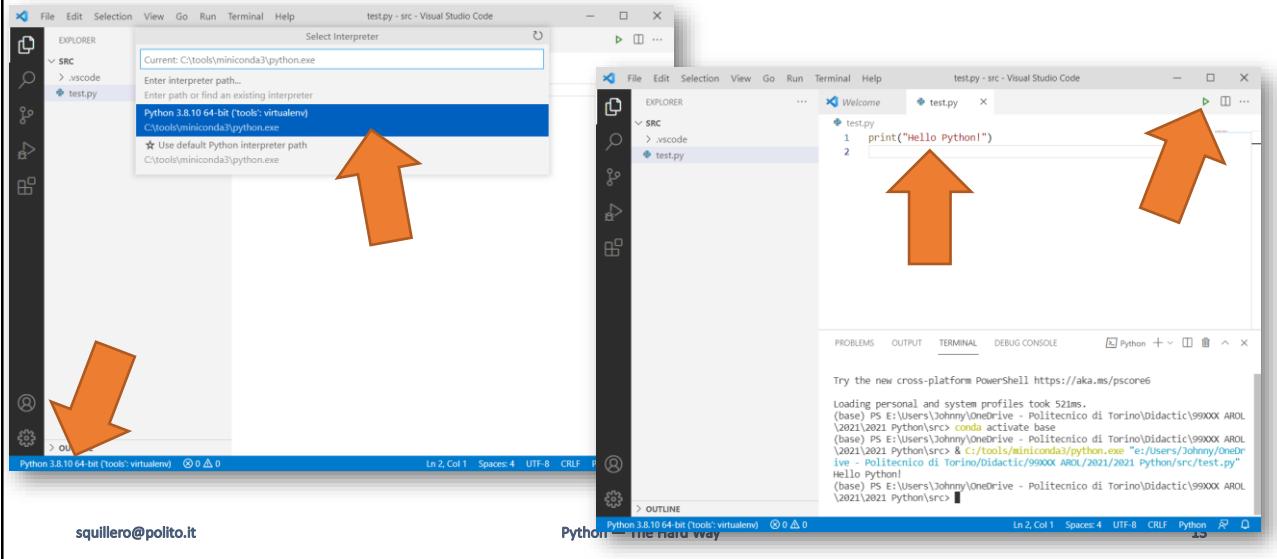
squillero@polito.it

Python — The Hard Way

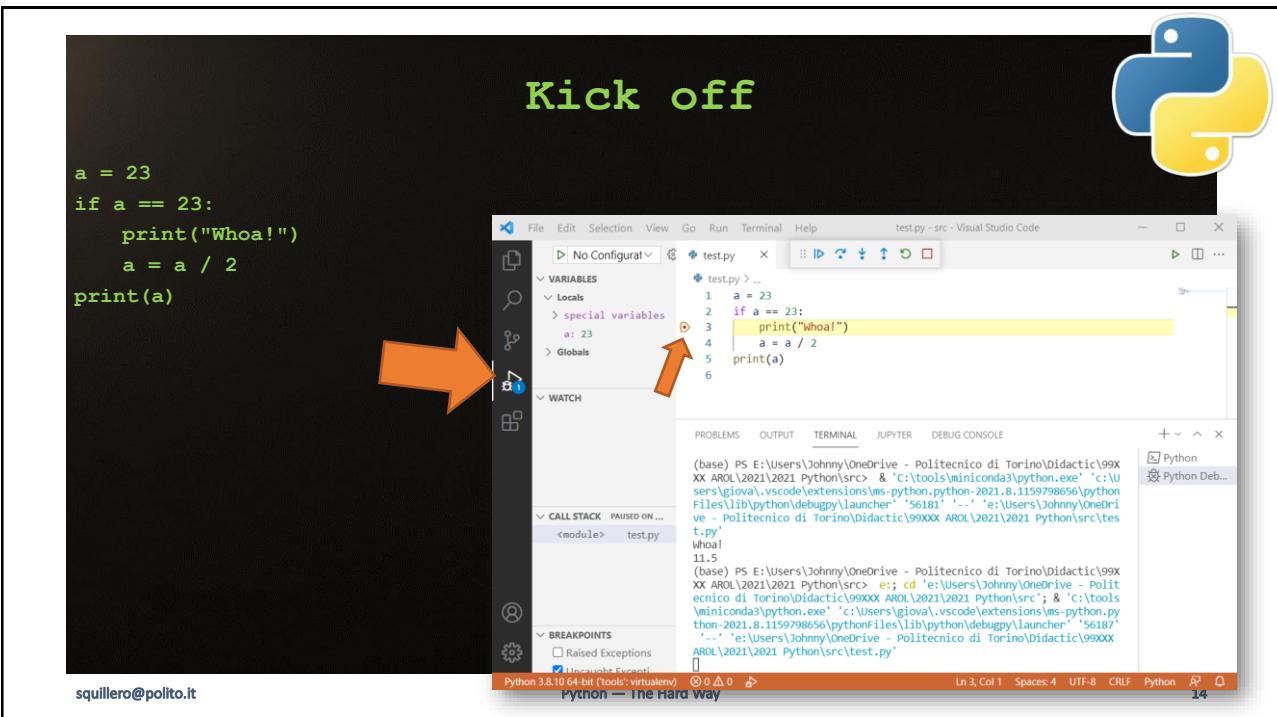
12

12

Set interpreter & Press play

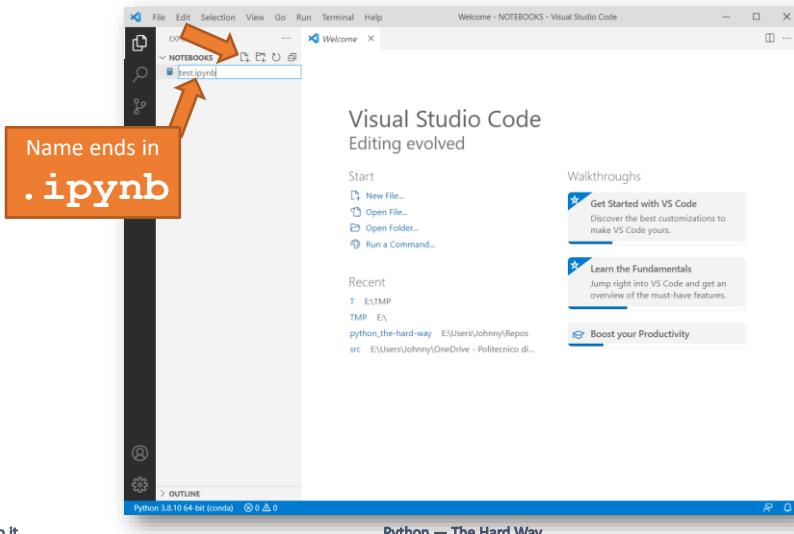


13



14

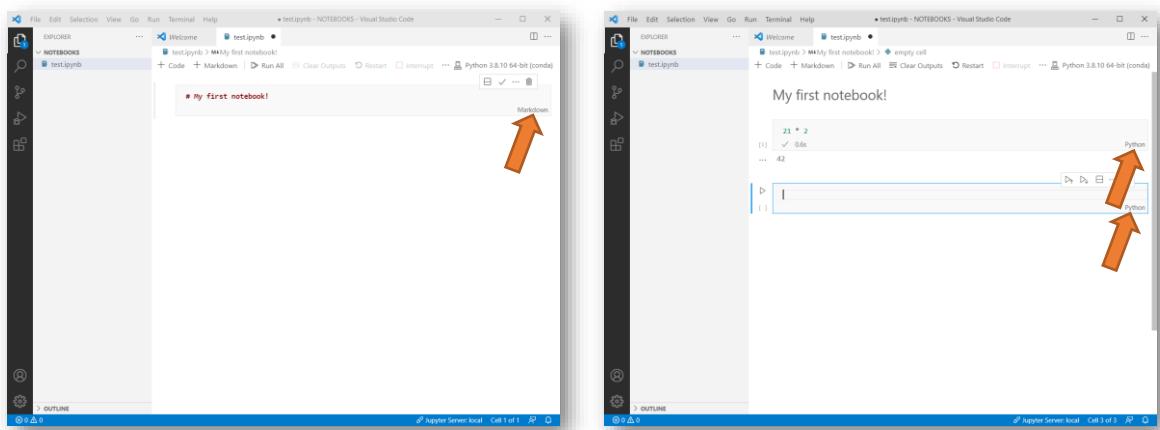
Jupyter Notebook



15

15

My first notebook



squillero@polito.it

Python — The Hard Way

16

16

Execution order...

```
[1] foo = 42
    ✓ 0.4s Python

[4] foo += 1
    bar = 23
    ✓ 0.3s Python

[3] bar += 10
    ✓ 0.2s Python

[5] print(foo, bar)
    ✓ 0.3s Python
... 44 23
```

squillero@polito.it

Python — The Hard Way

17

17

Python — The Hard Way

Data Types



18

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**

squillero@polito.it

Python — The Hard Way

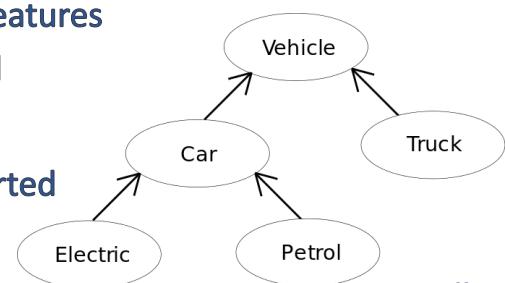
19

19

Object Oriented Paradigm (in 1 slide)

- An **object** contains both **data** and **code**
- An **objects** 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
- <code>xsize</code>
- <code>ysize</code>
- <code>label_text</code>
- <code>interested_listeners</code>
- <code>xposition</code>
- <code>yposition</code>
+ <code>draw()</code>
+ <code>press()</code>
+ <code>register_callback()</code>
+ <code>unregister_callback()</code>



squillero@polito.it

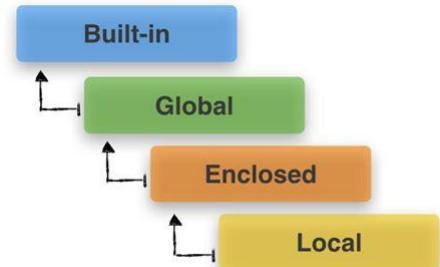
Python — The Hard Way

20

20

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 — The Hard Way

21

21

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 — The Hard Way

22

22

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 — The Hard Way

23

23

Standard Type Hierarchy

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



squillero@polito.it

Python — The Hard Way

24

24

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 — The Hard Way

25

25

Standard Type Hierarchy

- Mappings
 - Dictionaries (**dict**)

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

squillero@polito.it

Python — The Hard Way

26

26

Standard Type Hierarchy

- None (**NoneType**)

```
foo = None
```

squillero@polito.it

Python — The Hard Way

27

27

Standard Type Hierarchy

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



squillero@polito.it

Python — The Hard Way

28

28

Python — The Hard Way

Basic Syntax



29

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>

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 know what you are doing
 - `int_`

```
無 = 0
_ = 42
```

squillero@polito.it

Python — The Hard Way

31

31

Comments

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

```
# This is a comment
"""

This is a multi-line string,
and in certain contexts may be used as a comment
"""
```

squillero@polito.it

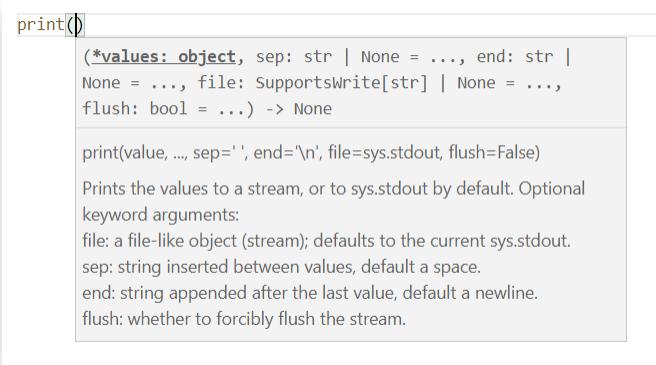
Python — The Hard Way

32

32

Basic I/O: `print`

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



squillero@polito.it

Python — The Hard Way

33

33

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='|')
```

squillero@polito.it

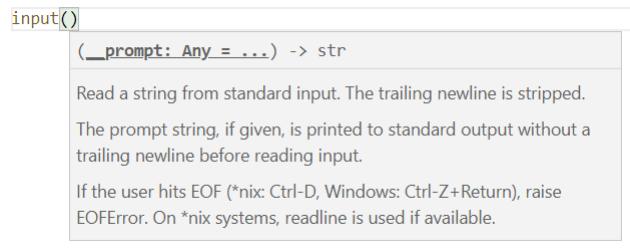
Python — The Hard Way

34

34

Basic I/O: `input`

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



squillero@polito.it

Python — The Hard Way

35

35

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 — The Hard Way

36

36

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 — The Hard Way

37

37

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)

squillero@polito.it

Python — The Hard Way

38

38

Numeric operations

Operation	Result	Notes
<code>x + y</code>	sum of x and y	
<code>x - y</code>	difference of x and y	
<code>x * y</code>	product of x and y	
<code>x / y</code>	quotient of x and y	
<code>x // y</code>	floored quotient of x and y	(1)
<code>x % y</code>	remainder of <code>x / y</code>	(2)
<code>-x</code>	x negated	
<code>+x</code>	x unchanged	
<code>abs(x)</code>	absolute value or magnitude of x	
<code>int(x)</code>	x converted to integer	(3)(6)
<code>float(x)</code>	x 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 $(x // y, x \% y)$	(2)
<code>pow(x, y)</code>	x to the power y	(5)
<code>x ** y</code>	x to the power y	(5)

squillero@polito.it

Python — The Hard Way

39

39

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 — The Hard Way

40

40

Bitwise operations

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

squillero@polito.it

Python — The Hard Way

41

41

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 — The Hard Way

42

42

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 — The Hard Way

43

43

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 — The Hard Way

44

44

Looping over multiple sequences

```
for a, b in zip('GIOVANNI', 'XYZ'):
    print(f"{a}:{b}")
✓ 0.3s
```

Python

G:X
I:Y
O:Z

squillero@polito.it

Python — The Hard Way

45

45

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 — The Hard Way

46

46

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)

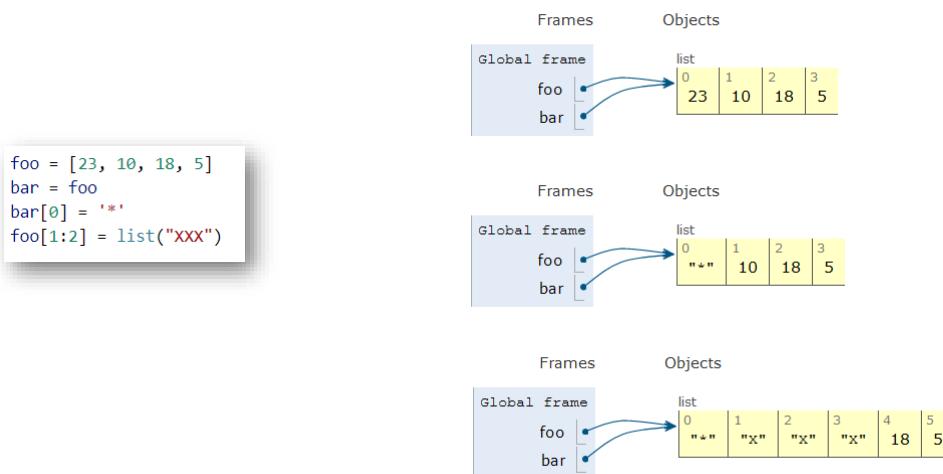
squillero@polito.it

Python — The Hard Way

47

47

Slices & Name binding



squillero@polito.it

Python — The Hard Way

48

48

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 — The Hard Way

49

49

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))
✓ 0.4s
[False, False, False, False, False, True]
True
False
```

Python

squillero@polito.it

Python — The Hard Way

50

50

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 — The Hard Way

51

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 — The Hard Way

52

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/library/string.html#format-spec>

squillero@polito.it

Python — The Hard Way

53

53

Notable `str` methods

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

Python

... ['Giovanni', 'Adolfo', 'Pietro', 'Pio']


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

Python

... ['Giovanni', 'Adolfo', 'Pietro', 'Pio']

- Caveat:

`"bar".foo()` vs. `str.foo("bar")`

squillero@polito.it

Python — The Hard Way

54

54

More notable `str` methods

```

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

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

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

```

squillero@polito.it

Python — The Hard Way

55

55

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 — The Hard Way

56

56

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 — The Hard Way

57

57

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 — The Hard Way

58

58

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 — The Hard Way

59

59

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 — The Hard Way

60

60

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 — The Hard Way

61

61

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:**

squillero@polito.it

Python — The Hard Way

62

62

While loop

- Vanilla, C-like **while**

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

squillero@polito.it

Python — The Hard Way

63

63

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 — The Hard Way

64

64

Python — The Hard Way

Functions



65

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

66

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 — The Hard Way

67

67

Named and Default Arguments

- Remember scope and name binding
- Arguments may be optional
- Named 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 — The Hard Way

68

68

True Pythonic Scripts

- Define all global ‘constants’ first, then functions
- Test __name__

```
GLOBAL_CONSTANT = 42

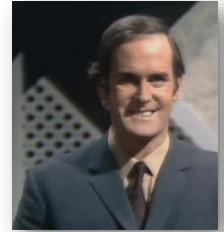
def foo():
    pass

def main():
    print(foo)
    pass

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

squillero@polito.it

Python — The Hard Way



69

69

Python — The Hard Way

List Comprehensions & Generators



70

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

squillero@polito.it

Python — The Hard Way

71

71

Generators

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

```
foo = (x**x for x in range(100_000))
foo
```

✓ 0.3s

Python

<generator object <genexpr> at 0x0000026D42A1C0B0>

```
for x in range(3):
    print(f'{next(foo):,}')
✓ 0.3s
```

1

1

4

Python

```
for x in range(3):
    print(f'{next(foo):,}')
✓ 0.4s
```

27

256

Python

squillero@polito.it

Python — The Hard Way

72

72

Generators

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

```
any([n % 23 == 0 for n in range(1, 1_000_000_000)])
```

✓ 63.8s

Python

True

```
any(n % 23 == 0 for n in range(1, 100_000_000))
```

✓ 0.3s

Python

True

squillero@polito.it

Python — The Hard Way

73

73

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 — The Hard Way

74

74

Python — The Hard Way

Modules



75

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
```

Python

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

Python

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

```
[12:07:40] INFO: Verbose information
[12:07:40] WARNING: Watch out, it looks important
[12:07:40] ERROR: We have a problem
[12:07:40] CRITICAL: Armageddon was yesterday, now we have a real problem...
```

squillero@polito.it

Python — The Hard Way

77

77

Notable Modules

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

```
import math

math.
```

0.4 **math.acos**

math.acosh
math.asin
math.asinh
math.atan
math.atan2
math.atanh
math.ceil
math.comb
math.copysign
math.cos
math.cosh

Python

squillero@polito.it

Python — The Hard Way

78

78

Notable Modules

- Common string operations and constants

The screenshot shows a code editor window with a Python script. The script starts with `import string` and then imports the `string` module. A tooltip or dropdown menu is open over the `string` import statement, displaying the list of items defined in the module's __all__ list. The items listed are: ascii_letters, ascii_lowercase, ascii_uppercase, capwords, digits, Formatter, hexdigits, octdigits, printable, punctuation, Template, and whitespace. The word 'ascii_letters' is highlighted in blue, indicating it is the currently selected item in the list.

```
import string

string.|
  [●] ascii_letters
  [●] ascii_lowercase
  [●] ascii_uppercase
  [●] capwords
  [●] digits
  [●] Formatter
  [●] hexdigits
  [●] octdigits
  [●] printable
  [●] punctuation
  [●] Template
  [●] whitespace
```

squillero@polito.it Python — The Hard Way 79

79

Notable Modules

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

The screenshot shows a code editor window with a Python script. The script imports the `pprint` module and defines a variable `numbers` as a list of sets. Each set contains integers representing powers of y for x from 1 to 10. The script then prints the `numbers` variable using `pprint`. The output is a nicely formatted list of sets, where each set is represented by curly braces containing the elements. The entire list is enclosed in square brackets.

```
from pprint import pprint

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

squillero@polito.it Python 80

80

Notable Modules

- Miscellaneous operating system interfaces

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

Python

os.

- abort
- access
- add_dll_directory
- altsep
- chdir
- chmod
- close
- closerange
- cpu_count
- curdir
- defpath
- device_encoding

Python — The Hard Way

squillero@polito.it

81

81

Notable Modules

- System-specific parameters and functions

```
import sys
sys.
✓ 0.5s
sys.addaudithook
```

Python

sys.

- 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

Python — The Hard Way

squillero@polito.it

82

82

Notable Modules

- Regular expression operations

```
import re

re.
  ✓ re.A
<module 're'>
    re.ASCII
    re.compile
    re.copyreg
    re.DEBUG
    re.DOTALL
    re.enum
    re.error
    re.escape
    re.findall
    re.finditer
    re.fullmatch
```

squillero@polito.it Python — The Hard Way 83

83

Notable Modules

- 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 — The Hard Way

84

84

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

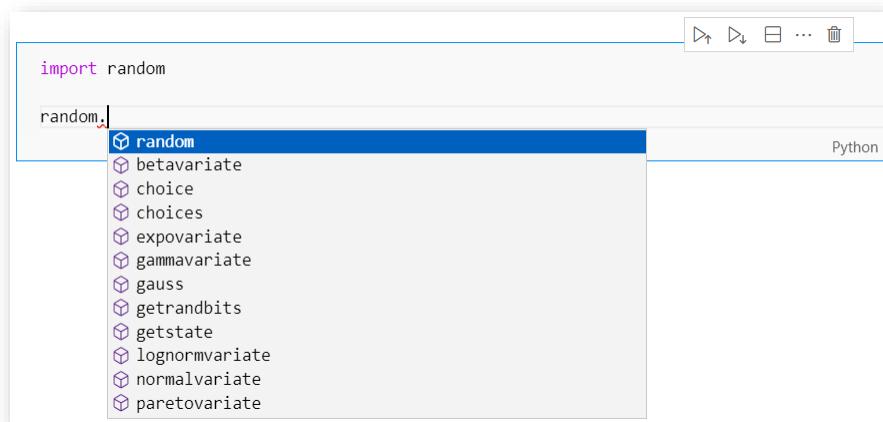
Python — The Hard Way

85

85

Notable Modules

- Generate pseudo-random numbers



The screenshot shows a code editor interface with a Python file open. The code starts with `import random`. Below it, the user has typed `random.` and is viewing the module's documentation. A tooltip window is displayed, titled "Python", listing various methods of the `random` module:

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

squillero@polito.it

Python — The Hard Way

86

86

Python — The Hard Way

Exceptions



87

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")
```

88

Notable Exceptions

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

squillero@polito.it

Python — The Hard Way

89

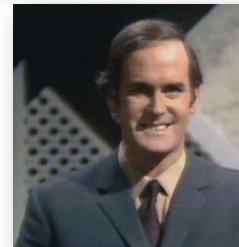
89

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



90

squillero@polito.it

Python — The Hard Way

90

Python — The Hard Way

i/o



91

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**'

92

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 — The Hard Way

95

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 — The Hard Way

96

96

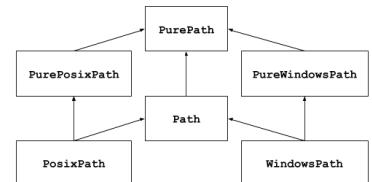
Paths

- 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 — The Hard Way

97



97

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 — The Hard Way

98



98

Read CSV

- As standard file

99

Read 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
```

100

Read Config

- 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 — The Hard Way

101