AI & Robotics

Introduction to Python: Part One

PXL AI & ROBOTICS LAB

Goals 1/2

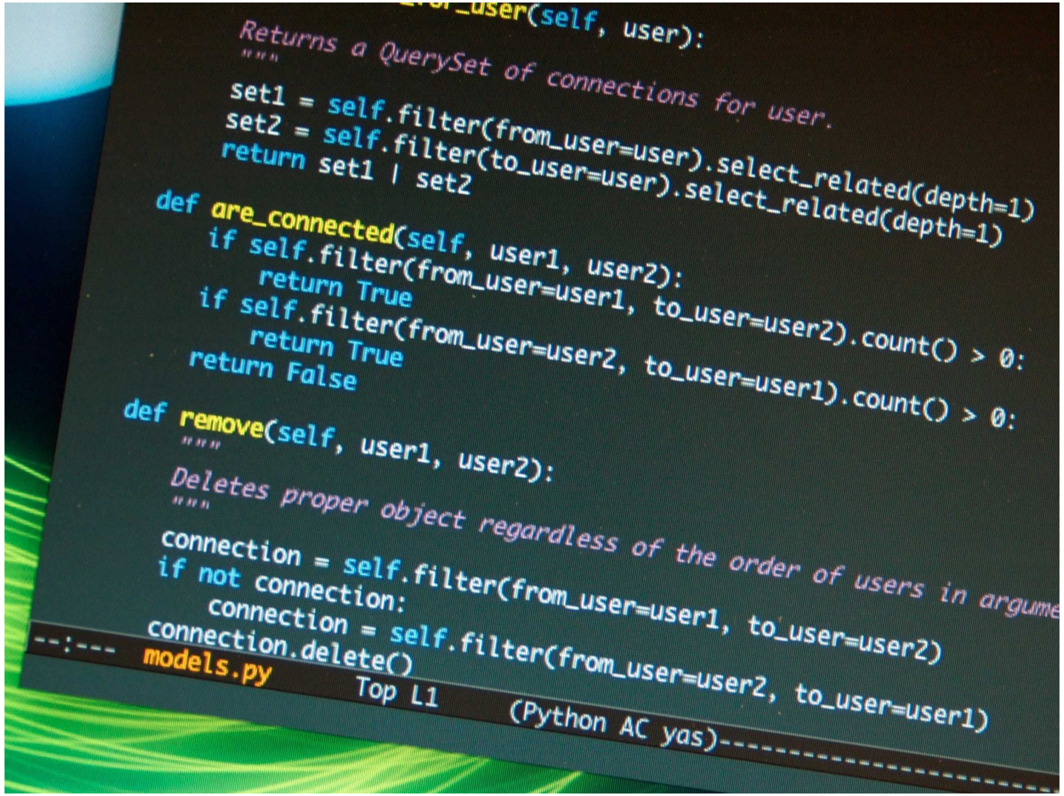
The **junior-colleague**

- can write fluent Python code.

Python

- **Very readable language**
- Control flow by **indentation**
  - No { } curly braces
  - You love it or hate it! *(I hope you will love it.)*
  - Clean code
- Dynamic Typing
  (No type definition needed.)

Control flow by indentation



```
            _user(self, user):
        Returns a QuerySet of connections for user.
        """
        set1 = self.filter(from_user=user).select_related(depth=1)
        set2 = self.filter(to_user=user).select_related(depth=1)
        return set1 | set2

    def are_connected(self, user1, user2):
        if self.filter(from_user=user1, to_user=user2).count() > 0:
            return True
        if self.filter(from_user=user2, to_user=user1).count() > 0:
            return True
        return False

    def remove(self, user1, user2):
        """
        Deletes proper object regardless of the order of users in argume
        """
        connection = self.filter(from_user=user1, to_user=user2)
        if not connection:
            connection = self.filter(from_user=user2, to_user=user1)
        connection.delete()
--:---  models.py       Top L1      (Python AC yas)-----------------------
```

The power of Python

- Multi-paradigm (All in one Python program / script!)
  (https://en.wikipedia.org/wiki/Programming_paradigm#Multi-paradigm)
  - **Imperative**
  - **Object oriented**
  - Functional
  - Reflective
  - . . .
- Cross platform
  (Linux, *BSD, Windows, Solaris, HP-UX, . . .)

The power of Python

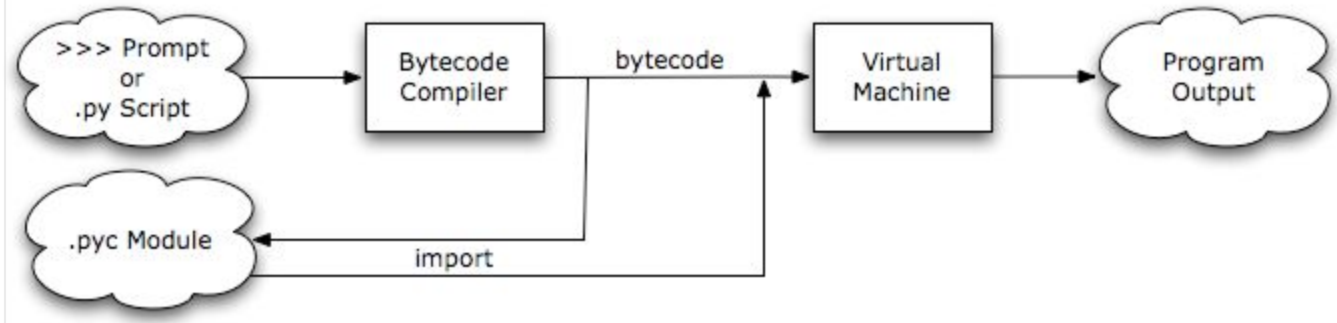
- Very extensive standard library
  - No need to reinvent the wheel
  - Powerful (A wide range of facilities)
- Extensively used at Google
  - Data synchronization between servers
  - Monitoring, logging and data collection
- Very popular
  - https://en.wikipedia.org/wiki/List_of_Python_software
  - http://www.tiobe.com/index.php/content/paperinfo/tpci/

Guido van Rossum

- Python BDFL
  (Benevolent Dictator For Life)
  (Permanent vacation since July 12, 2018)
- Dutch
- Currently: Dropbox
- @ Dropbox 50% for Python (Still?)
- Previously: Google
- Twitter: @gvanrossum

[MORE INFO]
Personal website of Guido van Rossum: https://www.python.org/~guido/
BDFL: https://en.wikipedia.org/wiki/Benevolent_dictator_for_life

Interpreter



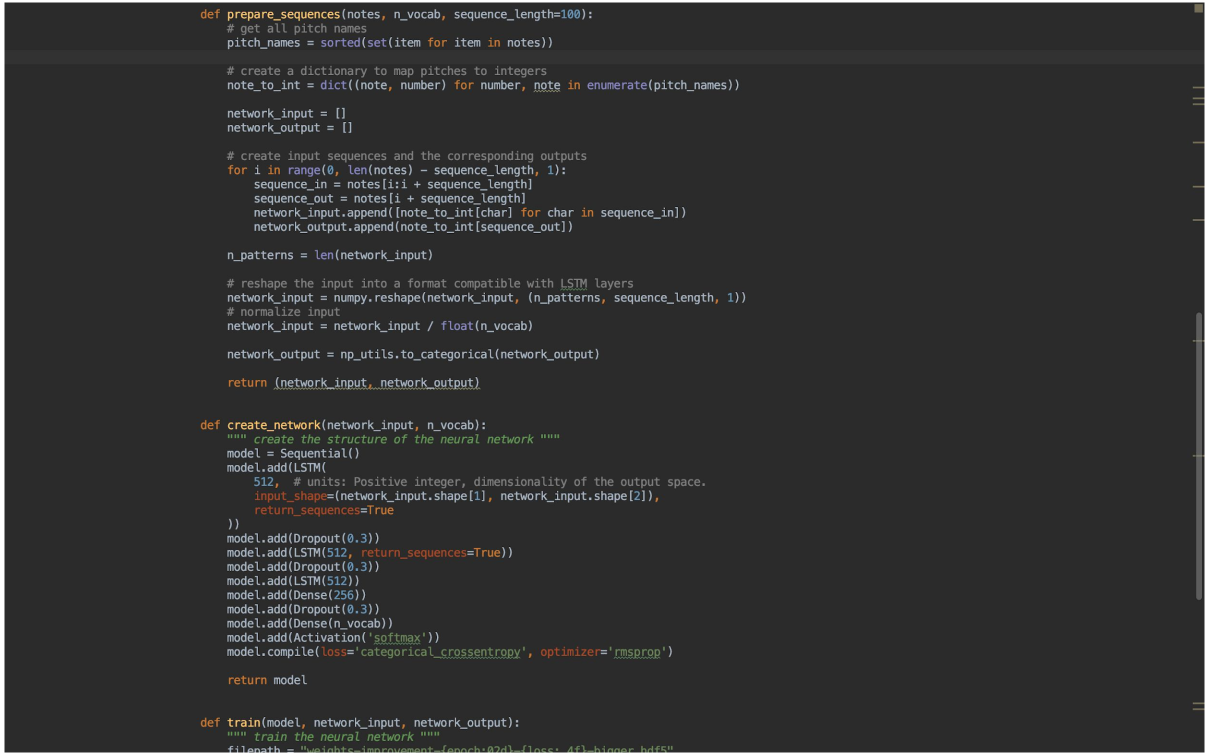
1. Parse source code
2. Create byte code
3. Import extra code, if necessary
4. Executes byte code

Interpreter

```
$ python
Python 3.7.1 (default, Oct 23 2018, 14:07:42)
[Clang 4.0.1 (tags/RELEASE_401/final)] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> quit()
$
```

On MacOS

PyCharm CE



```
def prepare_sequences(notes, n_vocab, sequence_length=100):
    # get all pitch names
    pitch_names = sorted(set(item for item in notes))

    # create a dictionary to map pitches to integers
    note_to_int = dict((note, number) for number, note in enumerate(pitch_names))

    network_input = []
    network_output = []

    # create input sequences and the corresponding outputs
    for i in range(0, len(notes) - sequence_length, 1):
        sequence_in = notes[i:i + sequence_length]
        sequence_out = notes[i + sequence_length]
        network_input.append([note_to_int[char] for char in sequence_in])
        network_output.append(note_to_int[sequence_out])

    n_patterns = len(network_input)

    # reshape the input into a format compatible with LSTM layers
    network_input = numpy.reshape(network_input, (n_patterns, sequence_length, 1))
    # normalize input
    network_input = network_input / float(n_vocab)

    network_output = np_utils.to_categorical(network_output)

    return (network_input, network_output)


def create_network(network_input, n_vocab):
    """ create the structure of the neural network """
    model = Sequential()
    model.add(LSTM(
        512,  # units: Positive integer, dimensionality of the output space.
        input_shape=(network_input.shape[1], network_input.shape[2]),
        return_sequences=True
    ))
    model.add(Dropout(0.3))
    model.add(LSTM(512, return_sequences=True))
    model.add(Dropout(0.3))
    model.add(LSTM(512))
    model.add(Dense(256))
    model.add(Dropout(0.3))
    model.add(Dense(n_vocab))
    model.add(Activation('softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='rmsprop')

    return model


def train(model, network_input, network_output):
    """ train the neural network """
    filepath = "weights-improvement-{epoch:02d}-{loss:.4f}-bigger.hdf5"
```

[SOURCE]
https://www.jetbrains.com/pycharm/download

Shebang!

- Must be the first line in a script
- Starts with: `#!`
- Followed by the interpreter’s full path

Bash: `#!/bin/bash`

Python: `#!/usr/bin/python`

Python: `#!/usr/bin/env python`

Identifiers

- Unique name
  - Start with letter or underscore
  - Followed by letters, numbers & underscores
- To refer to something
- Labels for:
  - Variables
  - Functions
  - Classes, instances
  - Modules
  - . . .

Keywords

and	del	from	not	while
as	elif	global	or	with
assert	else	if	pass	yield
break	except	import	print	
class	exec	in	raise	
continue	finally	is	return	
def	for	lambda	try	

These words can’t be used as identifiers!

(Do not use them to name variables, functions, . . .)

Data types: Few Numerical Types

Type	Information	Example
Integer	Implemented using C’s long type.	1027 211234
Long Integer	Size limited by the system.	567893L
Float	Implemented using C’s double type.	5.43 9483.123
bool	Boolean type	True of False

Data types: Some Sequence Types

Type	Information	Example
String	A list of characters. Immutable Enclosed in single or double quotes. Can span more than one line, with triple quotes.	"This is a string" """ A long, loooooong, looooooong string """
List	Can hold multiple types.	[1, 1.23, "Tim"] [1, 2, 3] [1.5, 2.7, 3.0] ["Tim", "Dupont", "Test"]
Tuple	Immutable list.	(1, 1.23, "Tim") (1.5, 2.7, 3.0) ("Tim", "Dupont", "Test")
Dictionary / Set	List of items, indexed with a key. Format: key:value, key:value, key:value, . . .	{"first":"Tim", "second":"Dupont"}

Strings: ' ' VS " "

Is there a difference? NO!

String literals can be enclosed in matching single quotes (') or double quotes ("). They can also be enclosed in matching groups of three single or double quotes (these are generally referred to as *triple-quoted strings*).

[**SOURCE**]
https://docs.python.org/2/reference/lexical_analysis.html#strings

Values in a Sequence

```
>>> "Tim"[0]
'T'
>>> "Tim"[1]
'i'
>>> "Tim"[2]
'm'
>>> "Tim"[3]
Traceback (most recent call last):
     File "<stdin>", line 1, in <module>   IndexError: string index
out of range
```

Value in a sequence

```
>>> ["Tim", "Dupont", "Test"][1]
'Dupont'
>>> (1, 2 , "Tim")[1]
2
>>> {"first":"Tim", "last":"Dupont"}[0]
Traceback (most recent call last):
     File "<stdin>", line 1, in <module>
KeyError: 0
>>> {"first":"Tim", "last":"Dupont"}["first"]
'Tim'
>>> {"first":"Tim", "last":"Dupont"}["last"]
'Dupont'
```

Range in a Sequence

```
>>> "Tim"[0:2]
'Ti'
>>> "Tim"[0:]
'Tim'
>>> "Tim"[1:2]
'i'
>>> "Tim"[1:3]
'im'
```

Variables

```
>>> name = "Tim"
>>> age = 32
>>> name
'Tim'
>>> age
32
>>>
```

- Simpler than:
  - Java
  - C
  - . . .
- Dynamic typing
  - Checked at run-time
  - Less type work

Duck Typing

```
>>> name = "Tim"
>>> age = 32
>>> name
'Tim'
>>> age
32
>>>
```

- name is a String
  (name = String value)
- age is a Integer
  (age = Integer value)

→ At run-time

→ Dynamically

→ **Each line**

Duck Typing

James Whitcomb Riley (may have coined the phrase) **:**

When I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird a duck.

Strongly Typed

```
>>> 3/3
1
>>> "3"/3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for /: 'str' and 'int'
>>>
```

- The type of a value doesn't suddenly change
- Only operations for the current type
- `/` is not applicable to strings types

  → Solution: casting. Here: `int(3) / 3`

Operators

Operator	Description	Use	Information
Addition	+	x + y	
Subtraction	-	x - y	
Multiplication	*	x * y	
Division	/	x / y	
Floor Division	//	x // y	9//2 = 4 and 9.0//2.0 = 4.0
Exponent	**	x ** y	x^y
Modulus	%	x % y	

Comparison Operators

Assume variable a holds 10 and variable b holds 20 . . .

Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	(a == b) is not true.
!=	If values of two operands are not equal, then condition becomes true.	(a != b) is true.
<>	If values of two operands are not equal, then condition becomes true.	(a <> b) is true. This is similar to != operator.
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

Assignment Operators

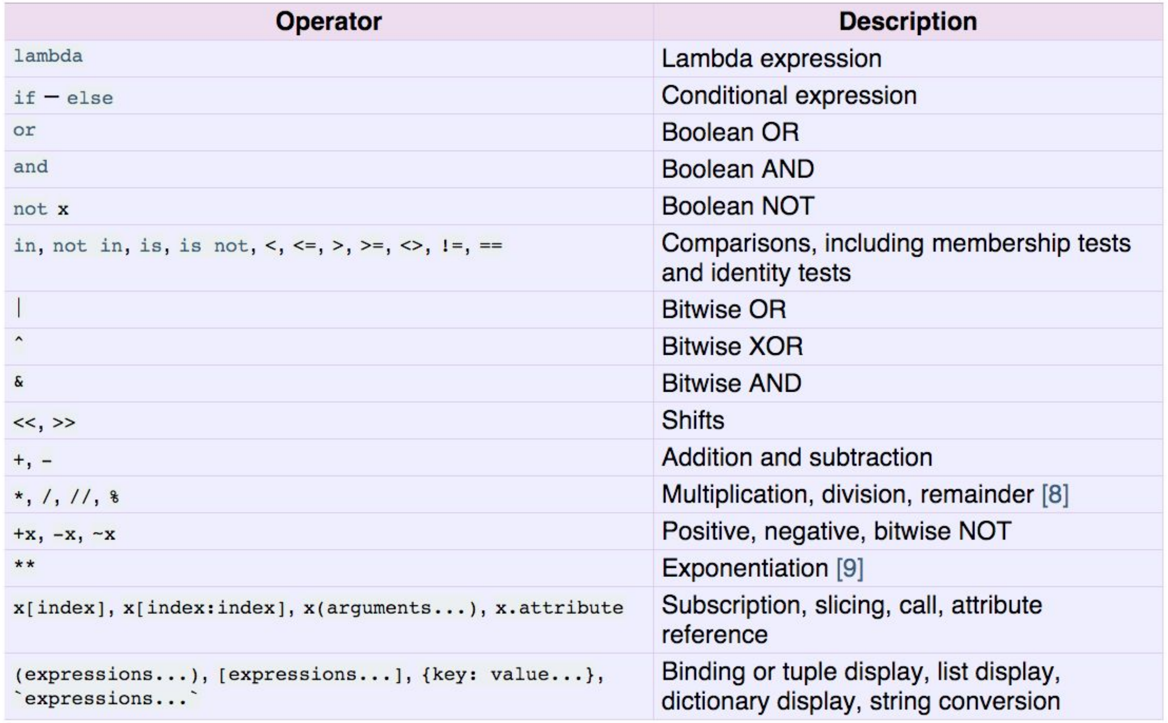
Assume variable a holds 10 and variable b holds 20 . . .

Operator	Description	Example
=	Assigns values from right side operands to left side operand	c = a + b assigns value of a + b into c
+= Add AND	It adds right operand to the left operand and assign the result to left operand	c += a is equivalent to c = c + a
-= Subtract AND	It subtracts right operand from the left operand and assign the result to left operand	c -= a is equivalent to c = c - a
*= Multiply AND	It multiplies right operand with the left operand and assign the result to left operand	c *= a is equivalent to c = c * a
/= Divide AND	It divides left operand with the right operand and assign the result to left operand	c /= a is equivalent to c = c / ac /= a is equivalent to c = c / a
%= Modulus AND	It takes modulus using two operands and assign the result to left operand	c %= a is equivalent to c = c % a
**= Exponent AND	Performs exponential (power) calculation on operators and assign value to the left operand	c **= a is equivalent to c = c ** a
//= Floor Division	It performs floor division on operators and assign value to the left operand	c //= a is equivalent to c = c // a

Boolean Operators

Operator	**Example**	**Information**
and	x and y	True if x and y are True
or	x or y	True if x or y are True
not	not x	True if x is False

Operator Precedence



Operator	Description
`lambda`	Lambda expression
`if – else`	Conditional expression
`or`	Boolean OR
`and`	Boolean AND
`not x`	Boolean NOT
`in`, `not in`, `is`, `is not`, `<`, `<=`, `>`, `>=`, `<>`, `!=`, `==`	Comparisons, including membership tests and identity tests
`|`	Bitwise OR
`^`	Bitwise XOR
`&`	Bitwise AND
`<<`, `>>`	Shifts
`+`, `-`	Addition and subtraction
`*`, `/`, `//`, `%`	Multiplication, division, remainder [8]
`+x`, `-x`, `~x`	Positive, negative, bitwise NOT
`**`	Exponentiation [9]
`x[index]`, `x[index:index]`, `x(arguments...)`, `x.attribute`	Subscription, slicing, call, attribute reference
`(expressions...)`, `[expressions...]`, `{key: value...}`, `` `expressions...` ``	Binding or tuple display, list display, dictionary display, string conversion

From lowest precedence (least binding) to highest precedence (most binding)
Operators in the same box have the same precedence.

Instructions

```
>>> 5 + \
... 5
10
>>> 5 + 5 # Gives 10
10
>>>
```

- 1 line = 1 instruction
- No “;”
- Multiple lines? $\rightarrow$ \
  (Like Bash)
- Comments: #
  (Like Bash)

Conversion Functions

Function	Description
int(x [,base])	Converts x to an integer. base specifies the base if x is a string.
long(x [,base])	Converts x to a long integer. base specifies the base if x is a string.
float(x)	Converts x to a floating-point number.
str(x)	Converts object x to a string representation.
tuple(s)	Converts s to a tuple.
list(s)	Converts s to a list.
set(s)	Converts s to a set.
dict(d)	Creates a dictionary. d must be a sequence of (key,value) tuples.

Print to the console

```
>>> name = "Tim"
>>> print(name)
Tim
>>> age = 35
>>> print(age)
35
>>> 'Name and age: ', name, " ", age
('Name and age: ', 'Tim', ' ', 35)
>>> print('Name and age: ' , name , " " , age)
Name and age:  Tim   35
```

User Input

```
>>> input()
This is input
'This is input'
>>> input('Input --> ')
Input -->
''

>>> input('Input --> ')
Input --> This is input
'This is input'
>>>
```

Sequence Length

```
>>> len("Tim")
3
>>> len((1,2,3,4,"Tim"))
5
>>>
```

- For a sequence
- So also for: tuple & dictionary

Range

```
>>> range(7)
[0, 1, 2, 3, 4, 5, 6]
>>> range(5, 10)
[5, 6, 7, 8, 9]
>>> range(3,7)
[3, 4, 5, 6]
>>> range(-1, -15, -5)
[-1, -6, -11]
```

- Result: list
- range(*start value*, [*stop value*], [*step*])

String Functions

`str.strip([chars])`	Strip (left and right) whitespaces or specified string	`” test ”.strip() → ’test’`
		`”test”.strip(”t”) → ’es’`

`str.lstrip([chars])`	Strip (only left) whitespaces or specified string	`”test”.lstrip(”t”) → ’est’`
`str.rstrip([chars])`	Strip (only right) whitespaces or specified string	`”test”.rstrip(”t”) → ’tes’`

`str.startswith(prefix[, start[, end]])`	True if string starts with prefix.	`”test”.startwith(”te”) → True`
`str.endswith(suffix[, start[, end]])`	True if string ends with suffix	`”test”.endswith(”st”) → True`

String Functions

str.find(sub[, start[, end]])	Returns lowest index of sub	`"test".find("s") → 2`
str.rfind(sub[, start[, end]])	Returns highest index of sub	`"test".rfind("t") → 3`

str.replace(old, new[, count])	Replaces old with new	`"test".replace("t", "f") → 'fesf'`
		`"test".replace("t", "f",1) → 'fest'`

str.lower()	Make lowercase	`"TIM".lower() → 'tim'`
str.upper()	Make uppercase	`"tim".upper() → 'TIM'`
str.swapcase()	Swaps case for each character	`"Tim".swapcase() → 'tIM'`

String Functions

`str.split([sep[, maxsplit]])`	Split string, left to right. Returns list.	`”Lode Tim”.split()` → `[’Lode’, ’Tim’]`
		`”1:2:3”.split(”:”, 1)` → `[’1’, ’2:3’]`

`str.rsplit([sep[, maxsplit]])`	Split string, right to left. Returns list.	`”1:2:3”.rsplit(”:”, 1)` → `[’1:2’, ’3’]`
`count(sub[, start[, end]])`	Count occurrences	`”Chris Tim”.count(’i’)` → `2`

String Functions

`str.isalnum()`	Alphanumeric?	`”Tim26”.isalnum() → True`
		`”Tim 26”.isalnum() → False`

`str.isalpha()`	[a-zA-Z]?	`”Tim”.isalpha() → True`
		`”Tim26”.isalpha() → False`

`str.isdigit()`	[0-9]?	`”26”.isdigit() → True`
		`”Tim26”.isdigit() → False`

`str.islower()`		`”tim”.islower() → True`
		`”Tim”.islower() → False`

`str.isupper()`		`”TIM”.isupper() → True`
		`”Tim”.isupper() → False`

`str.isspace()`		`” ”.isspace() → True`
		`”T ”.isspace()→ False`

Sequence Functions

```
>>> sequence = [1,2,3,4]
>>> len(sequence)
4
>>> sequence.reverse()
>>> sequence
[4, 3, 2, 1]
>>> sequence.append(5)
>>> sequence
[4, 3, 2, 1, 5]
>>> sequence.pop()
5
```

Sequence Functions

```
>>> sequence
[4, 3, 2, 1]
>>> sequence.remove(3)
>>> sequence
[4, 2, 1]
>>> sequence.sort()
>>> sequence
[1, 2, 4]
```

Dictionary Functions

```
>>> {"first":"Tim","last":"Dupont"}.keys()
['last', 'first’]
>>> {"first":"Tim","last":"Dupont"}.get("first")
'Tim'
>>> {"first":"Tim","last":"Dupont"}["first"]
'Tim'
>>> {"first":"Tim","last":"Dupont"}.values()
['Dupont', 'Tim']
>>> {"first":"Tim","last":"Dupont"}.items()
[('last', 'Dupont'), ('first', 'Tim')]
```

Dictionary Functions

```
>>> dict = {"first":"Tim","last":"Dupont"}
>>> dict
{'last': 'Dupont', 'first': 'Tim'}
>>> dict.clear()
>>> dict
{}
```

Control Flow

- Control structures
- Two types:
  1. Iteration: `for` and `while`
  2. Choice (decision making) : `if`
     (Python doesn’t have a switch!)
- First line ends with a “:”
  (Indentation, no braces, cleaner code.)

Iterations: for

```
>>> courses = ["Scripting", "Linux", "Project"]
>>> for course in courses:
... print(course)
File "<stdin>", line 2
    print(course)
        ^
IndentationError: expected an indented block
>>> for course in courses:
...    print(course)
...
Scripting
Linux
Project
>>>
```

Iterations: while

```
>>> max=3
>>> counter=0
>>> while counter < max:
...     print("max: ", max)
...     print("counter:", counter)
...     counter += 1
...
max:   3
counter: 0
max:   3
counter: 1
max:   3
counter: 2
```

Choice: if

```
>>> name = "Tim"
>>> if name == "Tim":
...   print("Hi Tim!")
... elif name == "Lode":
...   print("Hello Lode!")
... else:
...   print("I don't know you!")
... Hi Tim!
>>>
```

Functions

Format:

```
def function_name(parameter1, parameter2=default_value):
   <code block>
   return value # optional
```

- Dynamically typed, return type is optional!
- Parameters are variable names.
- Default value for parameters is possible.
  (Those parameters are optional.)

Functions

```
>>> staff_members = ["Lode", "Tim", "Steven"]
>>> def is_staff_member(name, staff=staff_members):
...     if name in staff:
...         return True
...     return False
...
>>> is_staff_member("Tim")
True
>>> is_staff_member("Tim", ["Lode"])
False
>>> is_staff_member("Tim", ("Lode", "Tim"))
True
```

Functions

```
>>> is_staff_member(name="Tim", staff=("Lode", "Tim"))
True
>>> is_staff_member(staff=("Lode", "Tim"), name="Jan")
False
>>> is_staff_member(staff=("Lode", "Tim"), name="Tim")
True
>>> is_staff_member(staff=["Lode", "Tim"], name="Tim")
True
>>>
```

The __main__ function

```
#!/usr/bin/env python

if __name__ == '__main__':
    print("Only when the file is executed.")
```

[INFO] The separate main function, in the example below, creates more clarity.

```
#!/usr/bin/env python

def main():
    print("Only when the file is executed.")

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

Open Files

Format:

```
open(filename[, mode[, buff_size]])
```

- First parameter: filename
- Second: mode (r (standard), w, a, r+)
- Third: Not needed in this course
- Returns: File object

File Object Functions

file.readline([size])	Reads one complete line and returns it as a string. Returns an empty string when EOF.
file.readlines([sizehint])	Reads until EOF. Returns a list strings.
file.write(str)	Writes str to file.
file.writelines(sequence)	Writes sequence to file.

Files: Example

```
inputFile  = open("inputFile.txt")
outputFile = open("outputFile.txt", "w")

for line in inputFile.readlines():
    outputFile.write(line)
```

Writing to Buffers

Always close them!

```
>>> file = open("fileName", "w")
>>> file.closed
False
>>> file.close()
>>> file.closed
True
```

Modules

- Highest abstraction level
- Every .py file
- Module name = filename
- Import:
  1. `import os`
  2. `import os, sys`
  3. `from os import getcwd`
  4. `import os as operatingSystem`

Module Locations

- The directory where the python interpreter has been started or where the execute .py file exists.
- The environment variable: PYTHONPATH. Which is a list of module directories.
- Operating system’s standard library directories.

PYTHONPATH

```
>>> import sys
>>> print(sys.path)
['', '/usr/lib/python2.7',
'/usr/lib/python2.7/plat-x86_64-linux-gnu',
'/usr/lib/python2.7/lib-tk', '/usr/lib/python2.7/lib-old',
'/usr/lib/python2.7/lib-dynload',
'/usr/local/lib/python2.7/dist-packages',
'/usr/lib/python2.7/dist-packages']
>>>
```

First Own Module

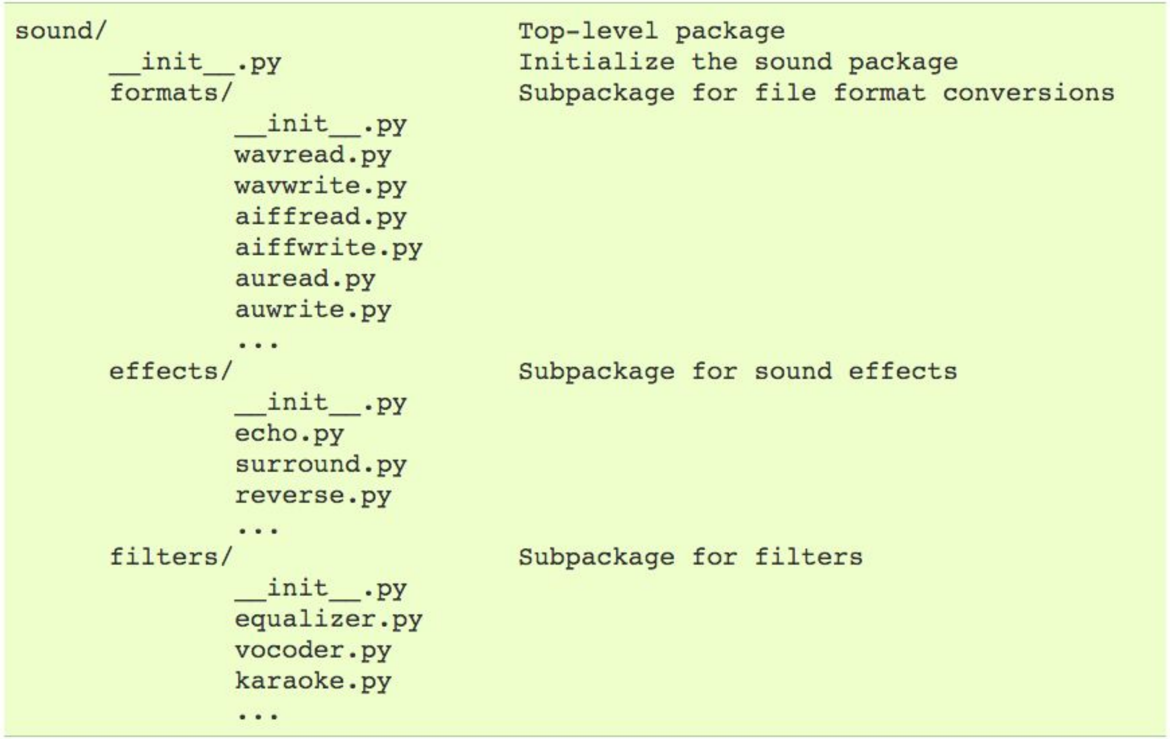
```
$ gedit foobar.py
   def foo():
       print("Foobar!")
$ python
>>> import foobar
>>> foobar.foo()
Foobar!
>>>
```

[INFO]
A .pyc file will be created.

Python Packages

- Contains a collections of modules
- Can have a hierarchy
- Directory with a `__init__.py` file
  (Prevents unintentionally hiding of valid modules.)
- Don’t forget: `PYTHONPATH`
- Module namespace structuring
- Less name collisions (= less worrying)
  (Modules with the same name exist in different packages.)

Example Python Package



```
sound/                          Top-level package
      __init__.py               Initialize the sound package
      formats/                  Subpackage for file format conversions
              __init__.py
              wavread.py
              wavwrite.py
              aiffread.py
              aiffwrite.py
              auread.py
              auwrite.py
              ...
      effects/                  Subpackage for sound effects
              __init__.py
              echo.py
              surround.py
              reverse.py
              ...
      filters/                  Subpackage for filters
              __init__.py
              equalizer.py
              vocoder.py
              karaoke.py
              ...
```

[SOURCE]
https://docs.python.org/2/tutorial/modules.html#packages

`__init__.py`

- Treat directory as a package container
- Can be an empty file
- Can execute initialization code
- Can set set the `__all__` variable
  (Not within the scope of this crash course.)

Import a Package

- `import sound.effects.echo`

  ```
  sound.effects.echo.echofilter(input, output, delay=0.7, atten=4)
  ```

- `from sound.effects import echo`

  ```
  echo.echofilter(input, output, delay=0.7, atten=4)
  ```

- `from sound.effects.echo import echofilter`

  ```
  echofilter(input, output, delay=0.7, atten=4)
  ```

Classes

- Definition: `class <ClassName>(parent):`
- Multiple classes in one Python module possible

```
class Person:
    def __init__(self, name):
        self.name = name
    def get_name(self):
        return self.name
    def set_name(self, name):
        self.name = name
```

Default Constructor

- Optional
- Default method: `__init__`

```
def __init__(self):
    #do something
```

Constructor With Parameters

```
def __init__(self, name, surname, age):
    self.name = strName
    self.surname = strSurname
    self.age = age
```

Constructors

- Only 1 constructor per class
- Multiple __init__ methods **not** allowed
- Dirty fix: use class methods. (Don’t do this.)

Properties

- No declaration without a value
- Declared in a method
- Use constructor for default value

```
class Person(object):

    def __init__(self):
        self.name = None
        self.surname = None
        self.age = 0
```

Methods

- A least one argument: **`self`**

```
def sleep(self):
    print("zzzzz")
```

- Return data

```
def get_name (self):
    return self.name
```

- Store data

```
def set_name(self, name):
    self.name = name
```

Destructor

- Method: __del__
- Automatically called
- Garbage collection

Class Variables

- Declared outside constructor or method
- Object independent

```
class Employee(object):
    employee_amount = 0
    def __init__(self):
        Employee.employee_amount += 1

employee_one = Employee()
employee_two = Employee()
print(Employee.employee_amount)
```

Class Inheritance

```
class Animal(object):
    def __init__(self, name, animal_type):
        self.name = name
        self.animal_type = animal_type
    def get_type(self):
        print(self.animal_type)

class Dog(Animal):
    def __init__(self, name):
        # Call parent constructor
        Animal.__init__(self, name, "dog")
    def bark(self):
        print("woof")
```

Method Overloading

- Not build in
- Possible via None values

```
class Employee(object):
    def set_stuff(self, name=None, surname=None):
        self.name = name
        self.surname = surname

employee = Employee()
employee.setStuff("Dupont")
print(employee.name, employee.surname)
```

Data hiding

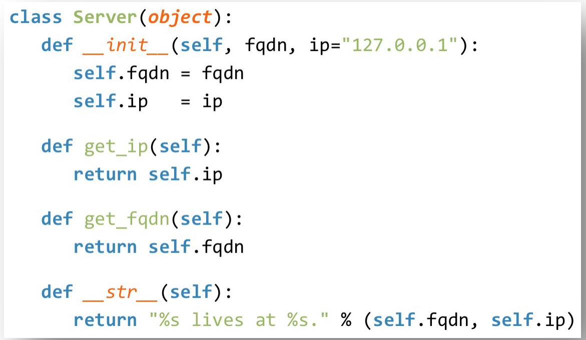
- No public/private/protected keywords
- Use common sense:
  Don’t access members directly
- Possible with double underscore

```
__hidden_var = 10
```

Classes & Inheritance

```
$ cat ServerComm.py

class Server(object):
   def __init__(self, fqdn, ip="127.0.0.1"):
      self.fqdn = fqdn
      self.ip   = ip

   def get_ip(self):
      return self.ip

   def get_fqdn(self):
      return self.fqdn

   def __str__(self):
      return "%s lives at %s." % (self.fqdn, self.ip)
```

Classes & Inheritance



```
class Server(object):
    def __init__(self, fqdn, ip="127.0.0.1"):
        self.fqdn = fqdn
        self.ip   = ip

    def get_ip(self):
        return self.ip

    def get_fqdn(self):
        return self.fqdn

    def __str__(self):
        return "%s lives at %s." % (self.fqdn, self.ip)
```

- *`object`*: Inheritance
- *`__init__`*: Constructor
- `self`: Part of the instance

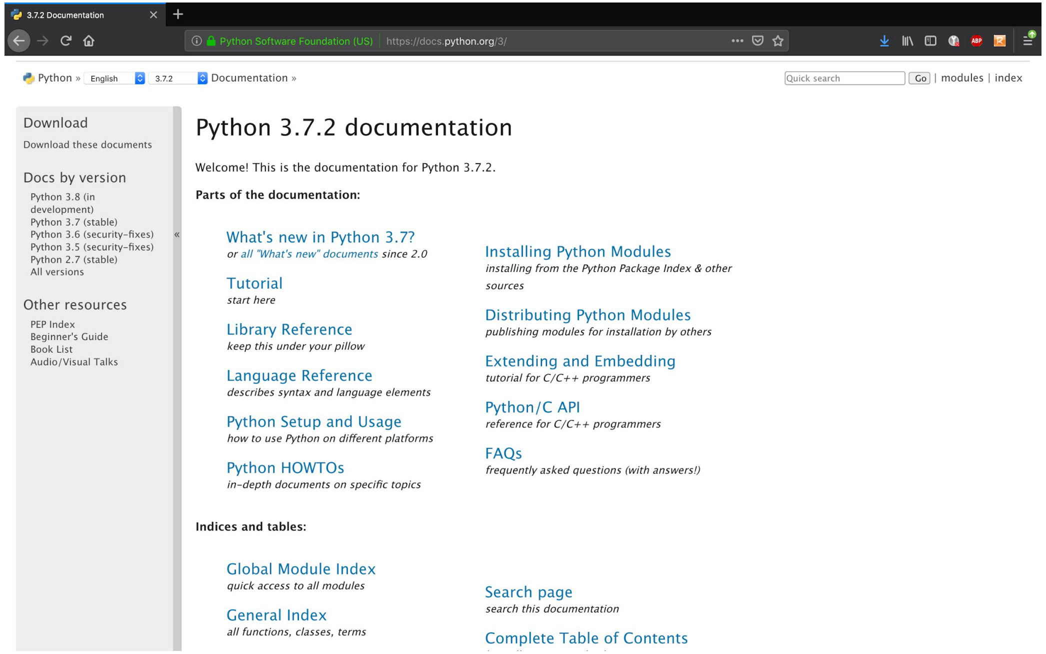
Classes & Inheritance

```
$ python
>>> from ServerComm import Server
>>> server = Server("localhost")
>>> print(server)
localhost lives at 127.0.0.1.
>>> server.get_ip()
'127.0.0.1’
>>> server.get_fqdn()
'localhost'
>>> quit()
$
```

Classes & Inheritance

```
$ tail -8 ServerComm.py

class FTPServer(Server):
   def __init__(self, fqdn, ip="127.0.0.1", port=21):
      Server.__init__(self, fqdn, ip)
      self.port = port

   def get_port(self):
      return self.port
$ python
>>> import ServerComm
>>> server = ServerComm.FTPServer("localhost")
>>> server.get_port()
21
>>>
```

Documentation



3.7.2 Documentation

Python Software Foundation (US) https://docs.python.org/3/

Python » English 3.7.2 Documentation »

Quick search Go | modules | index

Download

Download these documents

Docs by version

Python 3.8 (in development)
Python 3.7 (stable)
Python 3.6 (security-fixes)
Python 3.5 (security-fixes)
Python 2.7 (stable)
All versions

Other resources

PEP Index
Beginner's Guide
Book List
Audio/Visual Talks

Python 3.7.2 documentation

Welcome! This is the documentation for Python 3.7.2.

Parts of the documentation:

What's new in Python 3.7?
or all "What's new" documents since 2.0

Tutorial
start here

Library Reference
keep this under your pillow

Language Reference
describes syntax and language elements

Python Setup and Usage
how to use Python on different platforms

Python HOWTOs
in-depth documents on specific topics

Installing Python Modules
installing from the Python Package Index & other sources

Distributing Python Modules
publishing modules for installation by others

Extending and Embedding
tutorial for C/C++ programmers

Python/C API
reference for C/C++ programmers

FAQs
frequently asked questions (with answers!)

Indices and tables:

Global Module Index
quick access to all modules

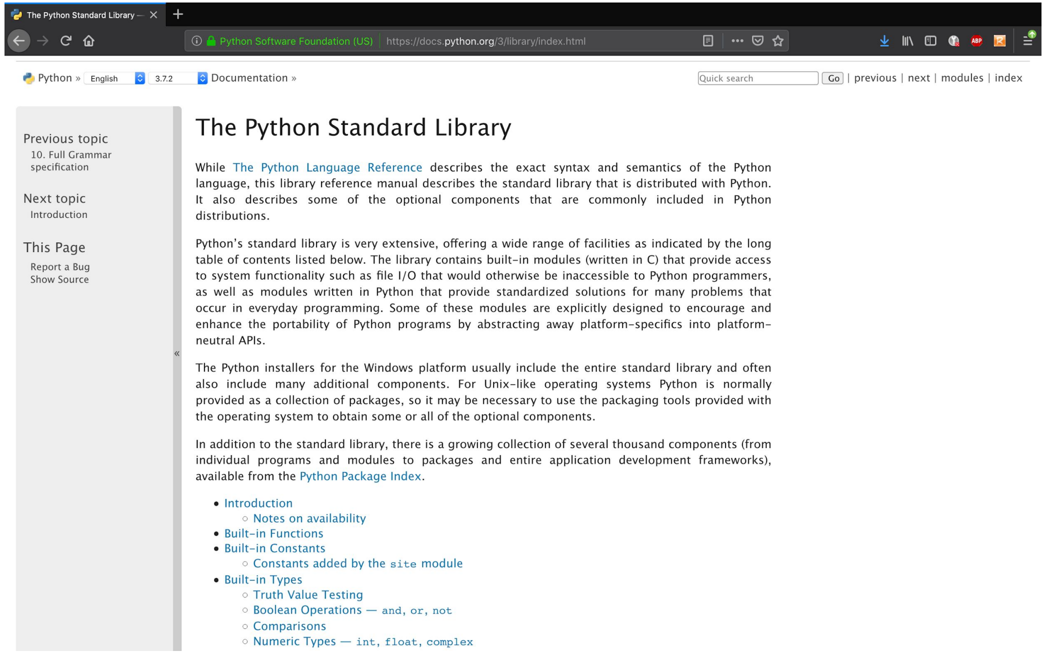
General Index
all functions, classes, terms

Search page
search this documentation

Complete Table of Contents

[SOURCE]
https://docs.python.org/3

Documentation: The Python Standard Library



The Python Standard Library —

Python Software Foundation (US) https://docs.python.org/3/library/index.html

Python » English 3.7.2 Documentation »

Quick search Go | previous | next | modules | index

Previous topic
10. Full Grammar specification

Next topic
Introduction

This Page
Report a Bug
Show Source

The Python Standard Library

While The Python Language Reference describes the exact syntax and semantics of the Python language, this library reference manual describes the standard library that is distributed with Python. It also describes some of the optional components that are commonly included in Python distributions.

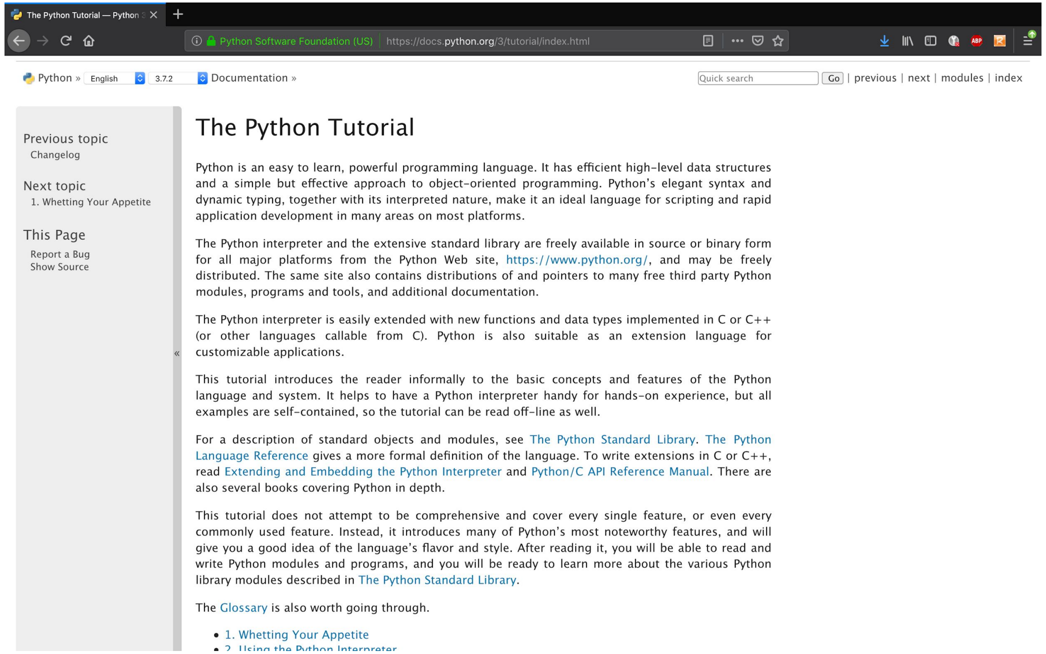
Python's standard library is very extensive, offering a wide range of facilities as indicated by the long table of contents listed below. The library contains built-in modules (written in C) that provide access to system functionality such as file I/O that would otherwise be inaccessible to Python programmers, as well as modules written in Python that provide standardized solutions for many problems that occur in everyday programming. Some of these modules are explicitly designed to encourage and enhance the portability of Python programs by abstracting away platform-specifics into platform-neutral APIs.

The Python installers for the Windows platform usually include the entire standard library and often also include many additional components. For Unix-like operating systems Python is normally provided as a collection of packages, so it may be necessary to use the packaging tools provided with the operating system to obtain some or all of the optional components.

In addition to the standard library, there is a growing collection of several thousand components (from individual programs and modules to packages and entire application development frameworks), available from the Python Package Index.

[**SOURCE**]
https://docs.python.org/3/library/index.html

Documentation: The Python Tutorial


The Python Tutorial — Python
Python Software Foundation (US)
https://docs.python.org/3/tutorial/index.html
Python » English 3.7.2 Documentation »
Quick search
Go
previous | next | modules | index
Previous topic
Changelog
Next topic
1. Whetting Your Appetite
This Page
Report a Bug
Show Source

The Python Tutorial

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

The Python interpreter and the extensive standard library are freely available in source or binary form for all major platforms from the Python Web site, https://www.python.org/, and may be freely distributed. The same site also contains distributions of and pointers to many free third party Python modules, programs and tools, and additional documentation.

The Python interpreter is easily extended with new functions and data types implemented in C or C++ (or other languages callable from C). Python is also suitable as an extension language for customizable applications.

This tutorial introduces the reader informally to the basic concepts and features of the Python language and system. It helps to have a Python interpreter handy for hands-on experience, but all examples are self-contained, so the tutorial can be read off-line as well.

For a description of standard objects and modules, see The Python Standard Library. The Python Language Reference gives a more formal definition of the language. To write extensions in C or C++, read Extending and Embedding the Python Interpreter and Python/C API Reference Manual. There are also several books covering Python in depth.

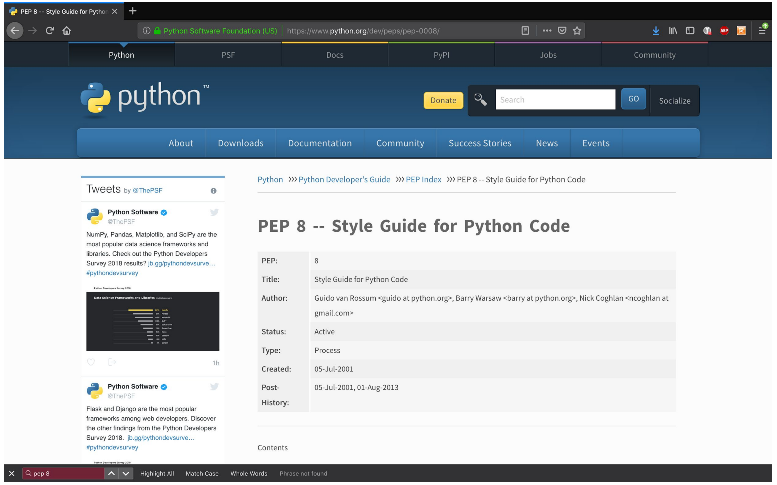
This tutorial does not attempt to be comprehensive and cover every single feature, or even every commonly used feature. Instead, it introduces many of Python's most noteworthy features, and will give you a good idea of the language's flavor and style. After reading it, you will be able to read and write Python modules and programs, and you will be ready to learn more about the various Python library modules described in The Python Standard Library.

The Glossary is also worth going through.

- 1. Whetting Your Appetite
- 2. Using the Python Interpreter

[**SOURCE**]
https://docs.python.org/3/tutorial/index.html

Style Guide for Python Code



Python Software @ThePSF

NumPy, Pandas, Matplotlib, and SciPy are the most popular data science frameworks and libraries. Check out the Python Developers Survey 2018 results? jb.gg/pythondevsurve… #pythondevsurvey

1h

Python Software @ThePSF

Flask and Django are the most popular frameworks among web developers. Discover the other findings from the Python Developers Survey 2018. jb.gg/pythondevsurve… #pythondevsurvey

Python ››› Python Developer's Guide ››› PEP Index ››› PEP 8 -- Style Guide for Python Code

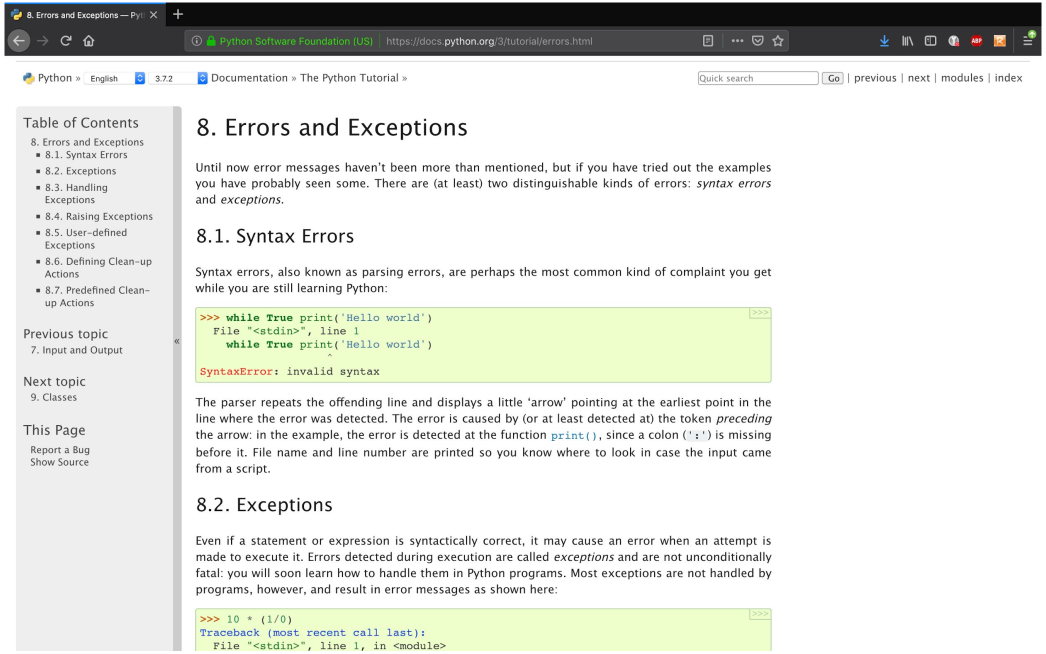
PEP 8 -- Style Guide for Python Code

PEP:	8
Title:	Style Guide for Python Code
Author:	Guido van Rossum <guido at python.org>, Barry Warsaw <barry at python.org>, Nick Coghlan <ncoghlan at gmail.com>
Status:	Active
Type:	Process
Created:	05-Jul-2001
Post-History:	05-Jul-2001, 01-Aug-2013

Contents

- PEP 8 -- Style Guide for Python Code (https://www.python.org/dev/peps/pep-0008/)
- Python Enhancement Proposal

Errors and Exceptions



8. Errors and Exceptions — Pyt

Python Software Foundation (US) https://docs.python.org/3/tutorial/errors.html

Python » English 3.7.2 Documentation » The Python Tutorial »

Quick search Go | previous | next | modules | index

Table of Contents

Previous topic
7. Input and Output

Next topic
9. Classes

This Page
Report a Bug
Show Source

8. Errors and Exceptions

Until now error messages haven't been more than mentioned, but if you have tried out the examples you have probably seen some. There are (at least) two distinguishable kinds of errors: *syntax errors* and *exceptions*.

8.1. Syntax Errors

Syntax errors, also known as parsing errors, are perhaps the most common kind of complaint you get while you are still learning Python:

```
>>> while True print('Hello world')
  File "<stdin>", line 1
    while True print('Hello world')
                   ^
SyntaxError: invalid syntax
```

The parser repeats the offending line and displays a little 'arrow' pointing at the earliest point in the line where the error was detected. The error is caused by (or at least detected at) the token *preceding* the arrow: in the example, the error is detected at the function `print()`, since a colon (`':'`) is missing before it. File name and line number are printed so you know where to look in case the input came from a script.

8.2. Exceptions

Even if a statement or expression is syntactically correct, it may cause an error when an attempt is made to execute it. Errors detected during execution are called *exceptions* and are not unconditionally fatal: you will soon learn how to handle them in Python programs. Most exceptions are not handled by programs, however, and result in error messages as shown here:

```
>>> 10 * (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
```

[**SOURCE**]
https://docs.python.org/3/tutorial/errors.html