1. Coding style: https://docs.python.org/3/tutorial/controlflow.html
2. Python is an interpreter language, and the indentation is Python’s way of grouping statements.
3. Variable \_

**>>>** tax = 12.5 / 100

**>>>** price = 100.50

**>>>** price \* tax

12.5625

**>>>** price + \_

113.0625

The variable \_ is read-only, the value is the last printed expression (price \* tax).

1. String

The strings can be enclosed in SINGLE quotes or DOUBLE quotes.

\ can be used to escape quotes (\” Hello)

Or **r** means raw strings, e.g.

**>>>** print('C:\some**\n**ame') *# here \n means newline!*

C:\some

ame

**>>>** print(**r**'C:\some\name') *# note the r before the quote*

C:\some\name

String literals can span multiple lines, one way is using TRIPLE-QUOTEs: “””…””” or ‘’’….’’’ ( \ will prevent the end of line)

print("""**\**

Usage: thingy [OPTIONS]

-h Display this usage message

-H hostname Hostname to connect to

""")

Usage: thingy [OPTIONS]

-h Display this usage message

-H hostname Hostname to connect to

Strings can be concatenated with the + operator, and repeated with \*:

**>>>** *# 3 times 'un', followed by 'ium'*

**>>>** 3 \* 'un' + 'ium'

'unununium'

Two or more string literals next to each other are automatically concatenated (not work with variables or expressions).

**>>>** 'Py' 'thon'

'Python'

**>>>** text = ('Put several strings within parentheses '

**...**  'to have them joined together.')

**>>>** text

'Put several strings within parentheses to have them joined together.'

Strings can be indexed, with the first character having index 0. Indices may also be negative numbers, to start counting from the right (-1, -0 is the same as 0).

Slicing is also supported, e.g. word[0:2], word[:2], word[2:]

+---+---+---+---+---+---+

| P | y | t | h | o | n |

+---+---+---+---+---+---+

0 1 2 3 4 5 6

-6 -5 -4 -3 -2 -1

Python strings cannot be changed.

Len(s)

1. List

Lists might contain items of different types, but usually the items all have the same type.

**>>>** squares = [1, 4, 9, 16, 25]

**>>>** squares

[1, 4, 9, 16, 25]

Like string, list can be indexed and sliced and support concatenation. All slicing operation return a new list.

Unlike strings, which are immutable, lists are a mutable type.

1. Slice operation will make a copy of original list

**>>>** *# Measure some strings:*

**...** words = ['cat', 'window', 'defenestrate']

**#** for w in words:

**>>> for** w **in** words[:]: *# Loop over a* ***slice copy*** *of the entire list.*

**...**  **if** len(w) > 6:

**...**  words.insert(0, w)

**...**

**>>>** words

['defenestrate', 'cat', 'window', 'defenestrate']

1. Range() : to iterate over a sequence of numbers, e.g. range(5), range(5,10), the combination of len(). Range()
2. a [try](https://docs.python.org/3/reference/compound_stmts.html#try) statement’s else clause runs when no exception occurs, and a loop’s else clause runs when no break occurs

**>>> for** n **in** range(2, 10):

**...**  **for** x **in** range(2, n):

**...**  **if** n % x == 0:

**...**  print(n, 'equals', x, '\*', n//x)

**...**  **break**

**...**  **else**:

**...**  *# loop fell through without finding a factor*

**...**  print(n, 'is a prime number')

**...**

2 is a prime number

3 is a prime number

4 equals 2 \* 2

5 is a prime number

6 equals 2 \* 3

7 is a prime number

8 equals 2 \* 4

9 equals 3 \* 3

1. Pass statement – does nothing. It can be used when a statement is required syntactically but the program requires no action

**Case 1:**

**>>> while** **True**:

**...**  **pass** *# Busy-wait for keyboard interrupt (Ctrl+C)*

**...**

Case 2:

**>>> class** **MyEmptyClass**:

**...**  **pass**

**...**

Case 3:

**>>> def** initlog(\*args):

**...**  **pass** *# Remember to implement this!*

**...**

1. The [return](https://docs.python.org/3/reference/simple_stmts.html#return) statement returns with a value from a function. [return](https://docs.python.org/3/reference/simple_stmts.html#return) without an expression argument returns None
2. The default values are evaluated at the point of function definition in the defining scope

i = 5

**def** f(arg=i):

print(arg)

i = 6

f()

The result is 5, not 6

1. **Important warning:** The default value is evaluated only once. This makes a difference when the default is a mutable object such as a list, dictionary, or instances of most classes.

**def** f(a, L=[]):

L.append(a)

**return** L

print(f(1))

print(f(2))

print(f(3))

[1]

[1, 2]

[1, 2, 3]

The default value will not be shared with the following codes:

**def** f(a, L=**None**):

**if** L **is** **None**:

L = []

L.append(a)

**return** L

1. Functions can also be called using [keyword arguments](https://docs.python.org/3/glossary.html#term-keyword-argument) of the form kwarg=value. keyword arguments must follow positional arguments
2. When a final formal parameter of the form \*\*name is present, it receives a dictionary
3. Unpacking argument lists

**>>>** list(range(3, 6)) *# normal call with separate arguments*

[3, 4, 5]

**>>>** args = [3, 6]

**>>>** list(range(\*args)) *# call with arguments unpacked from a list*

[3, 4, 5]

**>>> def** parrot(voltage, state='a stiff', action='voom'):

**...**  print("-- This parrot wouldn't", action, end=' ')

**...**  print("if you put", voltage, "volts through it.", end=' ')

**...**  print("E's", state, "!")

**...**

**>>>** d = {"voltage": "four million", "state": "bleedin' demised", "action": "VOOM"}

**>>>** parrot(\*\*d) # dictionaries can delivery keyword arguments # # with the \*\*-operator

-- This parrot wouldn't VOOM if you put four million volts through it. E's bleedin' demised !

1. Small anonymous functions can be created with the [lambda](https://docs.python.org/3/reference/expressions.html#lambda) keyword

**>>> def** make\_incrementor(n):

**...**  **return** **lambda** x: x + n

**...**

**>>>** f = make\_incrementor(42) # function f is created by lambda

**>>>** f(0)

42

**>>>** f(1)

43

pass a small function as an argument

**>>>** pairs = [(1, 'one'), (2, 'two'), (3, 'three'), (4, 'four')]

**>>>** pairs.sort(key=**lambda** pair: pair[1])

**>>>** pairs

[(4, 'four'), (1, 'one'), (3, 'three'), (2, 'two')]

1. Print documentation strings (\_\_doc\_\_)

**>>> def** my\_function():

**...**  *"""Do nothing, but document it.*

**...**

**...**  *No, really, it doesn't do anything.*

**...**  *"""*

**...**  **pass**

**...**

**>>>** print(my\_function.\_\_doc\_\_)

1. Annotations are stored in the \_\_annotations\_\_ attribute of the function as a dictionary and have no effect on any other part of the function. Return annotations are defined by a literal ->, followed by an expression

**>>> def** f(ham: str, eggs: str = 'eggs') -> str:

**...**  print("Annotations:", f.\_\_annotations\_\_)

**...**  print("Arguments:", ham, eggs)

**...**  **return** ham + ' and ' + eggs

**...**

**>>>** f('spam')

Annotations: {'ham': <class 'str'>, 'return': <class 'str'>, 'eggs': <class 'str'>}

Arguments: spam eggs

'spam and eggs'

1. Using Lists as Stacks , append(). Pop()

Using Lists as Queues,  use [collections.deque](https://docs.python.org/3/library/collections.html" \l "collections.deque" \o "collections.deque) .

**>>> from** **collections** **import** deque

**>>>** queue = deque(["Eric", "John", "Michael"])

**>>>** queue.append("Terry") *# Terry arrives*

**>>>** queue.append("Graham") *# Graham arrives*

**>>>** queue.popleft() *# The first to arrive now leaves*

'Eric'

**>>>** queue.popleft() *# The second to arrive now leaves*

'John'

**>>>** queue *# Remaining queue in order of arrival*

deque(['Michael', 'Terry', 'Graham'])

A list comprehension consists of brackets containing an expression followed by a [for](https://docs.python.org/3/reference/compound_stmts.html#for) clause, then zero or more [for](https://docs.python.org/3/reference/compound_stmts.html#for) or [if](https://docs.python.org/3/reference/compound_stmts.html#if) clauses.

squares = [x\*\*2 **for** x **in** range(10)]

==

**>>>** squares = []

**>>> for** x **in** range(10):

**...**  squares.append(x\*\*2)

**...**

**>>>** squares

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

==

squares = list(map(**lambda** x: x\*\*2, range(10)))

 transpose rows and columns:

**>>>** matrix = [

**...**  [1, 2, 3, 4],

**...**  [5, 6, 7, 8],

**...**  [9, 10, 11, 12],

**...** ]

**>>>** [[row[i] **for** row **in** matrix] **for** i **in** range(4)]

[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]

1. A tuple consists of a number of values separated by commas, it is immutable, but may be nested

**>>>** t = 12345, 54321, 'hello!'

**>>>** t[0]

12345

**>>>** t

(12345, 54321, 'hello!')

**>>>** *# Tuples may be nested:*

**...** u = t, (1, 2, 3, 4, 5)

# the elements are packed together in a tuple, and the reverse operation is possible

**>>>** x, y, z = t

1. A set is an unordered collection with no duplicate elements

Curly braces or the [set()](https://docs.python.org/3/library/stdtypes.html#set) function can be used to create sets. Note: to create an empty set you have to use set(), not {}; the latter creates an empty dictionary,

**>>>** basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}

**>>>** a = set('abracadabra')

# union, intersection , difference and symmetric difference

1. dictionaries are indexed by keys, which can be any immutable type

It is best to think of a dictionary as an unordered set of key: value pairs, with the requirement that the keys are unique (within one dictionary). A pair of braces creates an empty dictionary: {}

1. looping techniques <https://docs.python.org/3/tutorial/datastructures.html>

When looping through dictionaries, the key and corresponding value can be retrieved at the same time using the items() method

When looping through a sequence, the position index and corresponding value can be retrieved at the same time using the [enumerate()](https://docs.python.org/3/library/functions.html#enumerate) function

To loop over two or more sequences at the same time, the entries can be paired with the [zip()](https://docs.python.org/3/library/functions.html#zip) function.

To loop over a sequence in reverse, first specify the sequence in a forward direction and then call the [reversed()](https://docs.python.org/3/library/functions.html#reversed) function.

To loop over a sequence in sorted order, use the [sorted()](https://docs.python.org/3/library/functions.html#sorted) function which returns a new sorted list while leaving the source unaltered.

1. A module is a file containing Python definitions and statements. The file name is the module name with the suffix .py appended. the module’s name (as a string) is available as the value of the global variable \_\_name\_\_.

**>>> import** **fibo # import fibo module**

**>>>** fibo.\_\_name\_\_

'fibo'

# If you intend to use a function often you can assign it to a local name:

**>>>** fib = fibo.fib # fib is the function defined in fibo

**>>>** fib(500)

1 1 2 3 5 8 13 21 34 55 89 144 233 377

# Variant of import

**>>> from** **fibo** **import** fib, fib2

**>>> from** **fibo** **import** \*

1. Each module has its own private symbol table, which is used as the global symbol table by all functions defined in the module.
2. **Note:** For efficiency reasons, each module is only imported once per interpreter session. Therefore, if you change your modules, you must restart the interpreter – or, if it’s just one module you want to test interactively, use [importlib.reload()](https://docs.python.org/3/library/importlib.html" \l "importlib.reload" \o "importlib.reload), e.g. import importlib; importlib.reload(modulename).
3. Executing modules as scripts

python fibo.py <arguments> # run a python module

#

**if** \_\_name\_\_ == "\_\_main\_\_":

**import** **sys**

fib(int(sys.argv[1])) # by adding this code at the end of your module

the name of the main module is always "\_\_main\_\_",

1. When a module named spam is imported, the interpreter first searches for a built-in module with that name. If not found, it then searches for a file named spam.py in a list of directories given by the variable [sys.path](https://docs.python.org/3/library/sys.html" \l "sys.path" \o "sys.path). [sys.path](https://docs.python.org/3/library/sys.html" \l "sys.path" \o "sys.path) is initialized from these locations:
2. To speed up loading modules, Python caches the compiled version of each module in the \_\_pycache\_\_ directory under the name module.*version*.pyc, where the version encodes the format of the compiled file; it generally contains the Python version number.
3. The built-in function [dir()](https://docs.python.org/3/library/functions.html" \l "dir" \o "dir) is used to find out which names a module defines. It returns a sorted list of strings

**>>> import** **fibo**, **sys**

**>>>** dir(fibo)

['\_\_name\_\_', 'fib', 'fib2']

Without arguments, [dir()](https://docs.python.org/3/library/functions.html" \l "dir" \o "dir) lists the names you have defined currently:

**>>>** a = [1, 2, 3, 4, 5]

**>>> import** **fibo**

**>>>** fib = fibo.fib

**>>>** dir()

['\_\_builtins\_\_', '\_\_name\_\_', 'a', 'fib', 'fibo', 'sys']

[dir()](https://docs.python.org/3/library/functions.html#dir) does not list the names of built-in functions and variables. If you want a list of those, they are defined in the standard module [builtins](https://docs.python.org/3/library/builtins.html" \l "module-builtins" \o "builtins: The module that provides the built-in namespace.): (dir(builtins))

1. Packages are a way of structuring Python’s module namespace by using “dotted module names”. For example, the module name A.B designates a submodule named B in a package named A.

The \_\_init\_\_.py files are required to make Python treat the directories as containing packages; In the simplest case, \_\_init\_\_.py can just be an empty file, but it can also execute initialization code for the package or set the \_\_all\_\_ variable, described later.

1. [repr()](https://docs.python.org/3/library/functions.html" \l "repr" \o "repr) is meant to generate representations which can be read by the interpreter (or will force a [SyntaxError](https://docs.python.org/3/library/exceptions.html" \l "SyntaxError" \o "SyntaxError) if there is no equivalent syntax).  [str()](https://docs.python.org/3/library/stdtypes.html" \l "str" \o "str) function is meant to return representations of values which are fairly human-readable. For objects which don’t have a particular representation for human consumption, [str()](https://docs.python.org/3/library/stdtypes.html" \l "str" \o "str) will return the same value as [repr()](https://docs.python.org/3/library/functions.html" \l "repr" \o "repr). Many values, such as numbers or structures like lists and dictionaries, have the same representation using either function.
2. [open()](https://docs.python.org/3/library/functions.html#open) returns a [file object](https://docs.python.org/3/glossary.html#term-file-object), and is most commonly used with two arguments: open(filename, mode).

‘r ’ for read

‘w’ for writing

‘a’ opens the file for appending

‘r+’ opens the file for both reading and writing

‘b’ appended to the mode opens the file in binary mode.

**>>>** f = open('workfile', 'w')

# Alternative way

#  The advantage is that the file is properly closed after its suite finishes, even if an exception is raised at some point.

**>>> with** open('workfile') **as** f:

**...**  read\_data = f.read()

**>>>** f.closed

True

For reading lines from a file, you can loop over the file object. This is memory efficient, fast, and leads to simple code:

>>>

**>>> for** line **in** f:

**...**  print(line, end='')

# Alternative way

**>>>** f.readline()

1. Serializing: Convert JSON to string representation.
2. Deserializing : Reconstructing the data from string representation.
3. A finally clause is always executed before leaving the [try](https://docs.python.org/3/reference/compound_stmts.html#try) statement, whether an exception has occurred or not.
4. The [raise](https://docs.python.org/3/reference/simple_stmts.html#raise) statement allows the programmer to force a specified exception to occur.
5. The [try](https://docs.python.org/3/reference/compound_stmts.html#try) … [except](https://docs.python.org/3/reference/compound_stmts.html#except) statement has an optional else clause, which, when present, must follow all except clauses. It is useful for code that must be executed if the try clause does not raise an exception. The use of the [else](https://docs.python.org/3/reference/compound_stmts.html#else) clause is better than adding additional code to the [try](https://docs.python.org/3/reference/compound_stmts.html#try) clause because it avoids accidentally catching an exception that wasn’t raised by the code being protected by the [try](https://docs.python.org/3/reference/compound_stmts.html#try) … [except](https://docs.python.org/3/reference/compound_stmts.html#except) statement.

try:

res = this\_might\_cause\_exception()

# do\_something(res)

except Exception as e:

# do something useful here

else:

do\_something(res)

1. Classes provide a means of bundling data and functionality together. Creating a new class creates a new type of object, allowing new instances of that type to be made.
2. passing an object is cheap since only a pointer is passed by the implementation

>>> def scope\_test():

spam = "test spam"

def do\_local():

nonlocal spam # will change the spam value, unless the spam defined in do\_local() # # cannot change the spam defined in outer function scope\_test()

spam = "local spam"

do\_local()

print(spam)

>>> scope\_test()

local spam

>>> def scope\_test():

spam = "test spam"

def do\_local():

global spam # will not change the spam in scope of scope\_test(), but it change the spam # in global

spam = "local spam"

do\_local()

print(spam)

>>> scope\_test()

test spam

>>> print(spam)

local spam

1. When a class definition is entered, a new namespace is created, and used as the local scope — thus, all assignments to local variables go into this new namespace.
2. Class objects support two kinds of operations: attribute (data, function object) references and instantiation.
3. Attribute references use the standard syntax used for all attribute references in Python: obj.name.
4. Class instantiation uses function notation. Just pretend that the class object is a parameterless function that returns a new instance of the class.

x = MyClass() # create a new instance of the class and # #assigns this object to the local variable x.

The instantiation operation (“calling” a class object) creates an empty object. Many classes like to create objects with instances customized to a specific initial state. Therefore a class may define a special method named [\_\_init\_\_()](https://docs.python.org/3/reference/datamodel.html#object.__init__), like this:

**def** \_\_init\_\_(self): # it will be invoked automatically

self.data = []

A method is a function that “belongs to” an object. x.f is a valid method reference, since MyClass.f is a function, but x.i is not, since MyClass.i is not. But x.f is not the same thing as MyClass.f — it is a method object, not a function object.

1. Generally speaking, instance variables are for data unique to each instance and class variables are for attributes and methods shared by all instances of the class:

**class** **Dog**:

kind = 'canine' *# class variable shared by all instances*

**def** \_\_init\_\_(self, name):

self.name = name *# instance variable unique to each instance*

>>> d = Dog('Fido')

>>> e = Dog('Buddy')

>>> d.kind *# shared by all dogs*

'canine'

>>> e.kind *# shared by all dogs*

'canine'

>>> d.name *# unique to d*

'Fido'

>>> e.name *# unique to e*

'Buddy'

Data attributes override method attributes with the same name;

It is not necessary that the function definition is textually enclosed in the class definition: assigning a function object to a local variable in the class is also ok. For example:

*# Function defined outside the class*

**def** f1(self, x, y):

**return** min(x, x+y)

**class** **C**:

f = f1

**def** g(self):

**return** 'hello world'

h = g

1. Inheritance

**class** **DerivedClassName**(BaseClassName):

<statement-1>

.

.

.

<statement-N>

Multiple Inheritance

**class** **DerivedClassName**(Base1, Base2, Base3):

<statement-1>

.

.

.

<statement-N>

1. a name prefixed with an underscore (e.g. \_spam) should be treated as a non-public part of the API (whether it is a function, a method or a data member). It should be considered an implementation detail and subject to change without notice.
2. Brief Tour of the Standard Library