Python Syntax Cheat Sheet

★ Use REPL in the command line to test ideas and prototype functions by running `python3`
★ Run a script file containing Python statements by running `python3 my_script.py`

Arithmetic Operators

Addition	`i + 2`
Subtraction	`i - 2`
Multiplication	`i * 2`
Division	`i / 2`
Floor Division	`i // 2`
Modulus (Remainder)	`i % 2`

Some of these operators are overloaded for specific data types; you can add lists together, and strings support repetition with the multiplication operator. Most overloaded operators also support built-in assignment (e.g. `*=`).

Data Types

String: declared with either single or double enclosing quotes.	`S = 'World'` `type(s) #str`
Int: integers.	`X = 3` `type(X) #int`
Float: floating-point (decimal) numbers.	`Y = 3.14159` `type(Y) #float`
Boolean: true or false.	`X = False` `Y = True`
List: an array-like structure which uses zero-based indexing for data access.	`L = [1, 'a']` `L[0]  # 1`
Tuple: an array-like struct; cannot be modified.	`T = (1, 2, 3)` `T[1]  # 2`
Dictionary: a hashed key-value pair structure.	`D = {'a': 1}` `D['a']  # 1`
Set: a hashed structure where all data is unique and immutable.	`S = {1, 2}`
Nonetype: null	`X = None`

Looping Mechanisms

For loops support iterating over any *iterable*, which includes collections (dict, set, list) and strings.	`L = [1, 2, 3]`
	`for e in L:`
	`  print(e)`

While loops support continuous looping as long as a boolean condition is met; the loop terminates when the condition is False.	`i = 0`
	`while i < 10:`
	`  print(i)`
	`  i += 1`

Looping over a range of integers can be done using the `range()` function.	`for i in range(1, 11):` `  print(i)`

Conditionals

Comparison: >, <, >=, <=, !=, == are supported.	`X == X  # True` `X != X  # False`
Logical Operands: `and`, `or` and `not` are used instead of &&, `||`, and `!`	`if x>100 or stop:` `  print('Stop!')`
In: `in` tests for presence of LHS in RHS.	`1 in [1, 2]  # T` `3 in [1, 2]  # F`

Is: `is` and `is not` test whether variables point to the same object.	`'Hello' is None #F`
	`if x is y:`
	`  print('same')`

Truthiness: Non-empty variables are considered "truthy" and can be used in conditional statements. `None`, `False`, 0, empty strings, and empty collections are not "truthy".	`x = {}`
	`if not x:`
	`  print('empty')`
	`s = input()`
	`if s:`
	`  print(f'echo {s}')`
	`if not s:`
	`  print('empty')`

Built-in Functions

Modifying collections: List: `append`, `insert` Dict: `pop`, `[ ]` Set: `add`, `discard`	`L = [1, 2, 3]`
	`L.pop(0)  # 1`
	`L.pop()  # 3`

Length: Gives the length of the provided data.	`len('Hello')  #5` `len({1: 'a'})  #1`
Type: Get the datatype of an object.	`type([1]) #list` `type(1)  #int`
Enumerate: Get both an index/counter and value of an element of a collection.	`for i, v in enumerate(X):` `  print(i, v)`
User input: Get user input (via standard input).	`s = input()` `print(f'echo {s}')`

Datatype casting: Convert object types by using the desired conversion function.	`int('123')`
	`str(3.14159)`
	`list({1, 2, 3})`
	`tuple([6, 7, 8])`

Zip iterable items: Combine two collections into a single iterable with zip().	`nums = [1, 2, 3]`
	`strs = ('one', 'two', 'three')`
	`for num, text in zip(nums, strs):`
	`  print(num, text)`

Minimum & maximum values: Get a collection's extreme member.	`d = ['xylophone', 'apple', 'cat']`
	`min(d)  # apple`
	`max(d)  # xyloph.`

Sorting: Sort members of a collection. A copy of the sorted elements is returned in a list.	`nums = [2, 4, 1, 3]`
	`nums = sorted(nums)`
	`# [1, 2, 3, 4]`
	`nums = {3, 2, 1}`
	`for e in sorted(nums):`
	`  print(e)`

Exceptions

Exceptions in Python interrupt execution due to an unrecoverable state. It is useful to **catch** exceptions using a **try/catch block**, and exceptions can be **raised** by the program's author to indicate undesirable or invalid state of the program.

```
d = {'a': 1, 'b': 2}
try:
    num = d['c']
except KeyError:
    print('oops.')
```

Classes

Define the class using the `class` keyword.	`class Test:` `  # ...`

Class methods are indented under the class definition and declared using the `def` keyword.	`class Test:`
	`  def demo(self):`
	`  pass`

Class attributes (data and functions) are designated using a reference to the class's instance. Typically, we call this instance `self`. It is *always* the first parameter of class functions.	`class Test:`
	`  def demo(self):`
	`  self.hi = True`
	`  self.hi()`
	`  def hi(self):`
	`  print('hi')`

Instances are created using the class's name followed by parentheses.	`t = Test()`
	`t.demo()`
	`# prints 'hi'`

"Dunder" methods allow classes to implement behavior for built-in methods, like `len()`, operators, and indexing.

```
class Test:
  def __init__(self):  # constructor
    print('new!')
  def __len__(self):   # len(Test)
    return 42
  def __add__(self, rhs): # + oper.
    return 42 + len(rhs)
```