Cheatography

Python Essentials Cheat Sheet

by Taissir Boukrouba (taissir2002) via cheatography.com/193976/cs/40402/

Datatypes

Text	`str`
Numeric	`int` , `float`
Sequence	`list` , `tuple` , `range`
Mapping	`dict`
Set	`set`
Other	`bool` , `Nonetype` , `bytes`

Casting

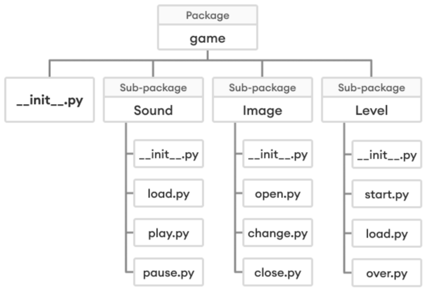
`int()`	converts into an integer	`int(2.8) = 2` `int("3") = 3`
`float()`	converts into float	`float(1) = 1.0` `float( " 3") = 3.0`
`str()`	converts into string	`str(3) = " 3"` `str(1.0) = " 1.0 "`

Casting is converting a datatype to another

Input & Output (I/O)

Output	we use the `print()` function	it has 3 main arguments which the `string` , the `seperator` and the `end` statement	`print( "Are you okay",e nd= " ?")` `print( " Hi", " How are you" ,"I missed you" ,se p="! !")`
Input	we use the `input()` function	the input function is used to take input from user and takes a `text` that is optional as argument	`num = input( 'Enter your age: ')`

Packages



A directory must contain a file named `init.py` in order for Python to consider it as a package. This file can be left empty but we generally place the initialisation code for that package in this file.

Operators

Logical (`and` , `or` , `not`)	used to check whether an expression is *True* or *False*	`a = 5`
		`b = 6`
		`print((a > 2) and (b >= 6))` > `True`

Operators (cont)

Identity (`is` , `is no t`)	used to check if two values are located on the same part of the memory	`x1 = 5`
		`y1 = 5`
		`print(x1 is not y1)`
		> `False`

Membership (`in` , `n ot in`)	used to test whether a value or variable is found in a sequence `(string, list, t uple, set, dictio nary)`	`x = 'Hello world'`
		`print( 'hello' not in x)`
		> `True`

for membership operators , in dictionaries it only checks the keys and not values

Module

- **Module** is a file that contains code to perform a specific task.

- A **module** may contain variables, functions, classes ...

- A collection of **modules** , can make what we call a **package**

As our program grows bigger, it may contain many lines of code. Instead of putting everything in a single file, we can use modules to separate codes in separate files as per their functionality. This makes our code organised and easier to maintain.

```
-----------------  example.py -----------------
def add(a, b):
 result = a + b
return result
----------------- main.py -----------------
import example
addition.add(4,5) # returns 9
```

List's Basic Operations

Accessing Lists	`list[i ndex]`	`languages = ["Py tho n", "Swift"]`
		`# access item at index 0`
		`print(languages[0])`

Slicing Lists	`list[f rom:to]`	`# List slicing in Python`
		`my_list = ['p',' r', 'o' ,'g ','r']`
		`# items from index 2 to index 4`
		`print(my_list[2:5])`

Adding one item at the end of list	`list.a ppe nd( item)`	`numbers = [21, 34, 54, 12]`
		`numbers.append(32)`

Adding All items of an iterable	`list1.e xt end (list2)`	`numbers = [1, 3, 5]`
		`even_numbers = [4, 6, 8]`
		`numbers.extend(even_numbers)`

List's Basic Operations (cont)

Adding one item at specific index	`list.i nse rt( ind ex, item)`	`numbers = [10, 30, 40]`
		`numbers.insert(1, 20)`

Changing item values	`list[i tem _index] = new_value`	`languages = ['Python', 'Swift', 'C++']`
		`# changing the third item to 'C'`
		`languages[2] = 'C'`

Removing one item of a list	`list.r emo ve( item)`	`languages = ['Python', 'Swift']`
		`# remove 'Python' from the list`
		`languages.remove('Python')`

Removing one or more items of a list	`del list[f rom:to]`	`del langua ges[1]`
		`del langua ge[0:2]`

Check if an item exists in a list	`item in list`	`languages = ['Python', 'Swift', 'C++']`
		`print('C' in languages)`
		`> False`

A list is a data structure that holds :
1) multiple data at once
2) of different data types (`str`,`int`,`float`)
3) can store duplicates

> we can create lists using brackets **[ ]** or the **list()** constructor

Other Lists Methods

Remove all items from a list	`list.c lear()`	`langua ges.cl ear()`

Return index of item	`list.i nde x(item)`	`animals = ['cat', 'dog', 'rabbit', 'horse']`
		`# get the index of 'dog'`
		`index = animal s.i nde x(' dog')`

Return length of a list	`len(list)`	`length (la ngu ages)`
		`> 3`

Return count of a specific item in a list	`list.c oun t(item)`	`numbers = [2, 3, 5, 2, 11, 2, 7]`
		`# check the count of 2`
		`count = number s.c ount(2)`

Sort a list (by default ascending)	`list.s ort (re ver se= fa lse)`	`vowels = ['e', 'a', 'u', 'o', 'i']`
		`vowels.sort(reverse=True)`
		`>['u', 'o', 'i', 'e', 'a']`

Other Lists Methods (cont)

Reverse items of list	`list.r eve rse()`	`prime_ numbers = [2, 3, 5, 7]`
		`# reverse the order of list elements`
		`prime_numbers.reverse()`

Copy a list	`list.c opy()`	`prime_ numbers = [2, 3, 5]`
		`# copying a list`
		`numbers = prime_ num ber s.c opy()`

List Comprehensions

Like there is a short way to write functions , there is a short one to also write lists and it's called **list comprehension**

Syntax : `[expre ssion for item in list]`

List comprehension is generally more compact and faster than normal functions and loops for creating list

Examples :

```
h_letters = [ letter for letter in 'human' ]
print(h_letters)
> ['h', 'u', 'm', 'a', 'n']
```

- We can add conditional to list comprehensions :

```
------ ----- example 01 -------------
number_list = [ x for x in range(20) if x % 2 == 0]
print(number_list)
 >[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
------ ----- example 02 -------------
num_list = [y for y in range(100) if y % 2 == 0 if y % 5 == 0]
print(num_list)
>[0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
```

Python Tuples

Accessing tuples	`tuple[ index]`	`letters = 'a','b ','c'`
		`letters[0]`

Slicing tuples	`tuple[ fro m:to]`	`letters = ('a',' b', 'c' ,'d ','e')`
		`letters[1:3]`

Return index of item	`tuple.i nd ex( item)`	`letters = ('a',' b', 'c' ,'d ','e')`
		`letters.index('a')`

Return count of a specific item	`tuple.c ou nt( item)`	`letters = ('a',' b', 'a' ,'d ','e')`
		`letters.count('a')`

Iterating over a tuple	`for item in tuple :`	`languages = ('Python', 'Swift', 'C++')`
		`# iterating through the tuple`
		`for language in languages:`
		` print( lan guage)`

Python Tuples (cont)

Check if a tuple element exists	`item in tuple`	`'C' in languages`

A **tuple** is a data structure that :
- holds multiple data at once
- of different types (`str`,`int`,`float`)
- can store duplicates
- is `immutable` so we cannot modify its items (this makes it faster to iterate over compared to lists) , meaning no delete or assignement operations

we can create lists using brackets `()` or just comma seperated value (meaning the `()` are optional) like follows :

```
first_ tuple = (1,2,3)
second _tuple = 1,2,3
```

Dictionaries

Accessing Items	`dictio nar y[key]`
	`dictio nar y.g et(key)`
		`countr y_c apitals = {`
		`"United States ": " Was hington D.C.",`
		`"Italy": " Rom e",`
		`"England": " Lon don " }`
		`print(country_capitals["United States"])`
		`> Washington D.C`

Removing Items	`del dictio nar y[key]`
	`dictio nar y.p op(key)`
		`sales = { 'apple': 2, 'orange': 3, 'grapes': 4 }`
		`popped_element = sales.p op ('a pple')`

Membership Test (keys only)	`key in dictionary`	`my_list = {1: " Hel lo", " Hi": 25, " How dy": 100}`
		`print(1 in my_list) -> True`
		`print("Howdy" not in my_list) -> False`
		`print("Hello" in my_list) -> False`

Dictionaries (cont)

Iterating Items	`for key,value in dictionary.items()`	`my_dict = {'apple': 1, 'banana': 2, 'orange': 3, 'grape': 4}`
		`for key, value in my_dict.items():`
		`print(f"Key: {key}, Value: {value}")`

A **dictionary** is a data structure and a collection that :
- allows us to store data in key-value pairs.
- **dictionary** keys must be immutable, such as tuples, strings, integers, etc meaning we cannot use mutable (changeable) objects such as lists as keys.
- **dictionary** values must be mutable of course

We create dictionaries by placing **key:value** pairs inside curly brackets **{ }**, separated by commas

Other Dictionary Methods

Update Items	`dictionary.update({key : new_value})`
	`dictionary.update( {new_key : new_value})`
		`d = {1: "one", 2: "three"}`
		`d1 = {2: "two"}`
		`# updates the value of key 2`
		`d.update(d1)`

Remove All Items	`dictionary.clear()`	`d.clear()`

Return All Keys	`dictionary.keys()`	`numbers = {1: 'one', 2: 'two', 3: 'three'}`
		`# extracts the keys of the dictionary`
		`dictionaryKeys = numbers.keys()`

Return All Values	`dictionary.values()`	`marks = {'Physics':67, 'Maths':87}`
		`print(marks.values())`

Return Items	`dictionary.items()`	`marks = {'Physics':67, 'Maths':87}`
		`print(marks.items())`
		`> dict_items([('Physics', 67), ('Maths', 87)])`

Other Dictionary Methods (cont)

Copy Dictionary	dictionary.copy()	original_marks = {'Physics':67, 'Maths':87}
		copied_marks = original_marks.copy()

Create Dictionary From Keys & Values	dict.fromkeys(keys,values)	keys = {'a', 'e', 'i', 'o', 'u' }
		value = [1]
		vowels = dict.fromkeys(keys, value)

Sets

Adding Items	set.add(item)	numbers = {21, 34, 54, 12}
		numbers.add(32)

Update Items	set.update(iterable)	companies = {'Lacoste', 'Ralph Lauren'}
		tech_companies = ['apple', 'google', 'apple']
		companies.update(tech_companies)
		print(companies)
		> {'google', 'apple', 'Lacoste', 'Ralph Lauren'}

Remove Items	set.discard(item)	removedValue = languages.discard('Java')

Checking if All Set Items Are True (or empty)	all(set) (stands for U or *)	L = [1, 3, 4, 5]
		print(all(L))
		> True

Checking if Any Set Items Are True	any(set) (stands for ∩ or +)	L = [1, 3, 4, 0]
		print(any(L))
		> True

Sets (cont)

Returning Enumerate Object	enumurate (iterable)	grocery = ['bread', 'milk', 'butter'] for count, item in enumerate (grocery): print(count, item) > 0 bread 1 milk 2 butter
Returning Length Of Set	len(set)	len(grocery)

Largest & Smallest item	max(set)
	min(set)
		numbers = [9, 34, 11, -4, 27]
		# find the maximum number
		max_number = max(numbers)

Sorting Set	sorted (set)	py_set = {'e', 'a', 'u', 'o', 'i'}
		print(sorted(py_set)
		> ['a', 'e', 'i', 'o', 'u']

Summing Set Items	sum(set)	marks = {65, 71, 68, 74, 61}
		# find sum of all marks
		total_marks = sum(marks)
		> 339

Iterate Over Set	for item in set :	fruits = {"Apple ", " Peach", " Mango"}
		# loop to access each fruits
		for fruit in fruits:
		print(fruit)

A **Set** is data structure that :

- Stores different data types
- Cannot have duplicates
- has **immutable** elements unlike lists and dictionaries

In Python, we create sets by placing all the elements inside curly braces **{ }** , separated by comma or using the **set()** constructor.

```
student_id = {112, 114, 116, 118, 115}
```

Set Operations

Union	set1.union(set2)
	set1 | set2
		A = {1, 3, 5}
		B = {0, 2, 4}
		print(A|B)
		> {0, 1, 2, 3, 4, 5}

Set Operations (cont)

Intersection	set1.intersection(set2)
	set1 & set2
		A = {1, 3, 5}
		B = {1, 2, 3}
		print(A & B)
		> {1, 3}

Difference	set1.difference(set2)
	set1 - set2
		A = {2, 3, 5}
		B = {1, 2, 6}
		print(A - B)
		> {3, 5}

Symmetric Difference	set1.symmetric_difference(set2)
	set1 ^ set2
		A = {2, 3, 5}
		B = {1, 2, 6}
		print(A ^ B)
		> {1, 3, 5, 6}

Python Strings

Accessing Strings	string [index]	greet = 'hello'
		print(greet[1])

Slicing Strings	string [from:to]	greet = 'hello'
		print(greet[0:2])

Comparing Two Strings	string1 == string2	str1 = " Hello, world! "
		str2 = "I love Python."
		print(str1 == str2)

Joining Strings	string1 + string2	str1 = " Hello, world! "
		str2 = "I love Python."
		print(str1 + str2)

String Length	len(string)	greet = 'hello'
		print(len(greet))

Formatting Strings (f-strings)	f"{string}"	print(f'{name} is from {country}')

Uppercase & Lowercase Strings	string.upper()
	string.lower()
		message = 'python is fun'
		print(message.upper())

Python Strings (cont)

Partitioning String Into Three Part Tuples	`string.partition(separator)`	`string = " Python is fun, isn't it`
		`print(string.partition('is'))`
		`>('Python ', 'is', " fun, isn't it`

Replacing Sub-String	`string.replace(old_substring,new_substring,occurrences`^{optional}`)`	`song = 'Let it be, let it be, let`
		`# replacing only two occurrences`
		`print(song.replace('let', " don't`

Return Index of Substring	`string.find(substring)`	`quote = 'Let it be, let it be, let`
		`# first occurance of 'let it'(case`
		`result = quote.find('let it')`

Remove Trailing Characters (By default removes whitespace)	`string.rstrip(substring`^{optional}`)`	`website = 'www.programiz.com/'`
		`print(website.rstrip('m/.'))`

Splitting Strings	`string.split(separator,maxsplit)`	`grocery = " Milk, Chicken, Bread,`
		`print(grocery.split(', ', 1))`
		`>["Milk","Chicken, Bread, Butter "`

Checking String Start	`string.startswith(substring)`	`text = " Python is easy to learn."`
		`result = text.startswith('is ea`
		`> False`

Python Strings (cont)

Advanced String Indexing	`string.index(substring , from, to $^{\text{optional}}$)	`sentence = 'Python programming is fun.'`
		`# Substring is searched in 'gramming is '`
		`print(sentence.index('g is', 10, -4))`

Python strings are immutable meaning we cannot change them , but we can assign its variable to another string which can do the job :

```
message = 'Hola Amigos'
message = 'Hello Friends'
```

Python Files

A **file** is a container in computer storage devices used for storing data.

When we want to read from or write to a file, we need to :

1- Open the file

2- Read or write in the file

3- Close the file

File Operations

Opening Files For Reading	`open(source,'r')`	`file1= open("test.txt",'r')`

Reading Files	`file.read()`	`read_content = file1.read()`
		`print(read_content)`

Closing Files	`file.close()`	`file1.close()`
Opening Files For Writing	`open(source,'w')`	`file2 = open("test.txt",'w')`
Writing in Files	`file.write(text)`	`file2.write( 'Programming is Fun.')`

Automatically Closing Files	`with open(source, mode) as filname :`
	`#instructions`
		`with open("test.txt", "r") as file1:`
		`read_content = file1.read()`
		`print(read_content)`

Directory Management

Get Current Working Directory	`os.getcwd()`	`import os`
		`print(os.getcwd())`
		`> /Users/taissirboukrouba/Data Science Cheat Sheet`

Directory Management (cont)

Changing Directory	`os.chd ir( new _di rec tory)`	`import os`
		`os.chdir('/Users/tayssirboukrouba/')`
		`print(os.getcwd())`

List Directories	`os.lis tdir()`	`import os`
		`os.chdir('/Users/tayssirboukrouba/')`
		`os.listdir()`

Making New Directory	`os.mkd ir( 'di r_n ame')`	`os.mkdir('test')`
		`os.listdir()`

Renaming Directory or File	`os.ren ame ('o ld_ dir ',' new _dir ')`	`import os`
		`os.listdir()`
		`os.rename('test','new_one')`
		`os.listdir()`

Removing Directories	`os.rem ove ('d ire ctory')`	`import os`
		`# delete " tes t.t xt" file`
		`os.remove("test.txt")`

A **Directory** is a collection of files and subdirectories.
A directory inside a directory is known as a **sub-directory** .

Python has the `os` module that provides us with many useful methods to work with directories (and files as well).

Conditionals

if	used to execute an instruction if a condition was true	`number = 0`
		`if number > 0:`
		` print( " Pos itive number " )`

elif	used to execute an instruction if the previous condition was not true and stands for *else if*	`elif number == 0:`
		` print( 'Zero')`

else	used to execute an instruction if all conditions were not true	`else print( "not positi ve")`

Loops

for	used mostly to loop through a sequence	`languages = ['Swift', 'Python', 'Go', 'JavaS cript']`
		`for language in languages:`
		`print( lan guage)`

Loops (cont)

while	used to loop through a statement while the condition is not met	`counter = 0`
		`while counter < 3:`
		`print( 'Inside loop')`
		`counter = counter + 1`
		`else  print( 'Inside else' )`

break	used to terminate the loop immediately when it is encountered	`for i in range(5):`
		` if i == 3:`
		` break`
		`print(i)`

continue	used to skip the current iteration of the loop and the control flow of the program goes to the next iteration	`for i in range(5):`
		` if i == 3:`
		` continue`
		`print(i)`

pass	null statement which can be used as a placeholder for future code	`n = 10`
		`if n > 10:`
		` pass`
		`print( 'He llo')`

Functions & Arguments

Syntax	`def functi on_ nam e(a rgu ments):`
	` # function body`
	`return`

Arguments with default values	`def add_nu mbers(a = 7,  b = 8):`

Arguments with keywords	`def displa y_i nfo (fi rst _name, last_n ame):`
	` print( 'First Name:', first_ name)`
	` print( 'Last Name:', last_name)`
	`display_info(last_name = 'Cartman', first_name = 'Eric')`

Functions & Arguments (cont)

Arbitrary Arguments	`def my_fun cti on( *kids):`
	`print("The youngest child is " + kids[2])`
	`my_function("Emil", " Tob ias ", " Lin us")`

If you do not know how many arguments that will be passed into your function, add a `*` before the parameter name in the function definition which will make the param an *arbitrary argument*

Variables Scopes

Local variable	a variables that is declared inside a function (cannot be accessed outside it)	`def greet():`
		`# local variable`
		` message = 'Hello'`
		` print( 'Lo cal', message)`
		`greet()`

Global variable	a variables that is declared outside a function (can be accessed outside or inside it)	`# declare global variable`
		`message = 'Hello'`
		`def greet():`
		` # declare local variable`
		` print( 'Lo cal', message)`
		`greet()`
		`print('Global', message)`

we can use the `global` keyword when we are inside a function , and we want to read and write a global variable inside a function.

Lambda Functions

Syntax	`lambda arguments : expression`

Example	`greet_user = lambda name : print( 'Hey,', name)`
	`greet_ use r(' Del ilah')`
	`> Hey, Delilah`

Lambda functions are also called anonymous functions because they have no name

Python OOP

Object	it's a collection of **data** (variables) and **methods** (functions).	`class Bike:`
		`#Attributes with default values :`
		`name = ""`
		`gear = 0`

Python OOP (cont)

Class	it is a **blueprint** or an example (sample) of that object	`bike1 = Bike()`

Accessing Class Attributes Using Objects	We use the **"."** notation to access the attributes of a class	`# modify the name attribute`
		`bike1.name = " Mou ntain Bike"`
		`# access the gear attribute`
		`bike1.gear`

Class Methods	A Python Function defined inside a class is called a **method**.	`class Room:`
		` length = 0.0`
		` breadth = 0.0`
		`# method to calculate area`
		`  def calcul ate _ar ea( self):`
		`   print( "Area of Room =", self.l ength * self.b read th)`

Constructors	We can initialise class using **`__init__()`** function	`class Bike:`
		`# constr uctor function`
		` def __init __( self, name = ""):`
		`  self.name = name`
		`bike1 = Bike()`
		`bike1 = Bike("M ountain Bike")`

Exception Handeling

`try-except` **Statement**	`try:`
	` numerator = 10`
	` denomi nator = 0`
	` result = numerator/denominator`
	` print(result)`
	`except:`
	` print( " Error: Denomi nator cannot be 0.")`
	`# Output: Error: Denomi nator cannot be 0.`

Exception Handeling (cont)

Catching Specific Exceptions

```
try:
 even_numbers = [2,4,6,8]
 print(even_numbers[5])
except ZeroDivisionError:
 print( " Denominator cannot be 0.")
except IndexError:
 print( " Index Out of Bound.")
# Output: Index Out of Bound
```

`try-else` Statement

```
# program to print the reciprocal of even numbers
try:
 num = int(input ("Enter a number: "))
 assert num % 2 == 0
except:
 print( "Not an even number!")
else:
 reciprocal = 1/num
 print( reciprocal)
```

`try-finally` Statement

```
try:
 numerator = 10
 denominator = 0
 result = numerator/denominator
 print( result)
except:
 print( " Error: Denominator cannot be 0.")
finally:
 print( "This is finally block." )
```

Exceptions can terminate the program's execution , that's why it is important to handle them

*when an exception occurs, the rest of the code inside the**try** block is skipped. If none of the statements in the **try** block generates an exception, the **except** block is skipped.*

In Python, the **finally** block is always executed no matter whether there is an exception or not.

Python Exceptions

`SyntaxError`	Raised when there is a syntax error in the code, such as incorrect indentation, invalid syntax, or mismatched parentheses.

Python Exceptions (cont)

`Indent ati onError`	A specific type of SyntaxError that occurs when there are problems with the indentation of the code.
`NameError`	Raised when a variable or name is used before it is defined.
`TypeError`	Occurs when an operation or function is applied to an object of an inappropriate type.
`ValueError`	Raised when a function receives an argument of the correct data type but an inappropriate value
`ZeroDi vis ion Error`	Occurs when attempting to divide by zero
`IndexError`	Raised when trying to access an index that is out of range for a list, tuple, or string.
`KeyError`	Raised when trying to access a non-existent key in a dictionary.
`Attrib ute Error`	Raised when an attribute or method is not found for an object.
`Import Error`	Occurs when a module cannot be imported.
`Assert ion Error`	Raised when an assert statement fails.
`Overfl owError`	Raised when the result of an arithmetic operation is too large to be represented.
`Memory Error`	Occurs when the Python interpreter cannot allocate enough memory for an object.
`Runtim eError`	A generic error that is raised when no specific exception applies.

An **exception** is an unexpected event (error) that occurs during program execution , for example :

```
divide _by _zero = 7 / 0
```

The above code causes an exception as it is not possible to divide a number by 0.