Python Cheatsheet

(Tested with Python 3.8)

The Python Cheatsheet is designed in an easy to understand flow with examples to learn the basics of Python. It can also be considered as a guide to Getting Started with Python. It covers the basic usage of sequences, binary sequences, list comprehension, for loop, while loop, if/else statement with examples. At the end, it provides the link to download the PDF having all the sections of the Python Cheat Sheet.

0 - Installers

Python 3.9 On macOS, Python 3.9 On Windows, Python 3.9 On Ubuntu, Python 3.8 On Windows, Python 3.8 On Ubuntu, PyCharm on Windows, PyCharm on Ubuntu, Visual Studio Code on Windows, Visual Studio Code on Ubuntu, Eclipse on Windows, and Eclipse on Ubuntu

1 - Data Types

```
# Base Data Types
Numeric: int, float, complex
Boolean: bool

# Container Data Types
Text: str
Sequence: list, tuple, range
Mapping: dict
Set: set, frozenset
Binary Sequence: bytes, bytearray, memoryview

# Data Examples
int: 12321, float: 12.342, complex: 2 + 12j
bool: True
str: "Hello Python"
list: ["apple", "banana", "orange"], tuple: ("apple", "banana", "orange"),
range: range(20)
dict: {"name" : "Rock", "age" : 18}
set: {"apple", "banana", "orange"}, frozenset: frozenset({"apple", "banana",
"orange"})
bytes: b"ABC", bytearray: bytearray(8), memoryview: memoryview(bytes(8))

# Data Range
int: unbounded
float: 2.2250738585072014e-308 to 1.7976931348623157e+308
```

2 - Containers (__contains__)

```
# Containers having method __contains__
str: "Hello Python"
list: ["apple", "banana", "orange"],
tuple: ("apple", "banana", "orange"),
range: range(20)
dict: {"name" : "Rock", "age" : 18}
set: {"apple", "banana", "orange"},
frozenset: frozenset({"apple", "banana", "orange"})
bytes: b"ABC"
bytearray: bytearray(8)

# Immutable: str, tuple, frozenset, bytes
# Access the contained objects
# Iterate over contained objects

# Supports in operator
tempStr = "Hello Python"
print( "llo" in tempStr )
# Output: True

# Test containers
from collections.abc import Container
tempStr = "Hello Python"
print( isinstance(tempStr, Container) )
# Output: True
```

3 - Keywords & Identifiers

```
# Keywords are reserved words in Python
# Keywords are case sensitive
# Keywords: if, else, elif, for, while, in, as, is, and, or, not, with, from,
break, pass, continue, return
# Keywords: class, finally, try, assert, def, nonlocal, del, yield
# Keywords: import, global, lambda, async, except, await, raise, None, True,
False

# Identifier can be used as name of variables, classes, or functions

# Identifier Rules
# Can have a to z, A to Z, 0 to 9, or _ underscore
# Cannot start with a digit
# Cannot use special characters including !, @, #, $, %
# Keywords cannot be used as Identifier
# Can be of any length
# Identifiers are case sensitive
# Identifier Examples
# Correct: alpha, _bravo, alpha12, alpha_bravo, alpha_123,
this_is_a_long_name
# Correct: alpha, Alpha, aLpha, and alphA : all are different
# Wrong: 12alpha, @alpha, al!pha, else, while
```

4 - Arithmetic Operators

```
+ : Addition: a + b, a + b + c
- : Subtraction: a - b, a + b - c
* : Multiplication: a * b, a * (b + c)
/ : Division: a / b, ( a + b ) / c
% : Modulus: a % b
** : Exponentiation: a ** b
// : Floor division: a // b
```

5 - Assignment Operators

```
= : Simple: a = b
+= : Increment: a += b ~ a = a + b
-= : Decrement: a -= b ~ a = a - b
*= : Multiplication: a *= b ~ a = a * b
/= : Division: a /= b ~ a = a / b
%= : Modulus: a %= b ~ a = a % b
**= : Exponentiation:a **= b ~ a = a ** b
//= : Floor division: a //= b ~ a = a // b
```

6 - Relational Operators

```
== : Equal: a == b
!= : Not equal: a != b
> : Greater than: a > b
< : Less than: a < b
>= : Greater than or equal to: a >= b
<= : Less than or equal to: a <= b
```

7 - Logical Operators

```
and : x == 25 and y == 35
or : x == 25 or y == 35
```

```
 not : not(x == 25 and y == 35)
```

8 - Identity & Membership Operators

```
# Identity Operators - Tests for same objects
 is : x is y
 is not : x is not y

# Membership Operators - Tests for sequence in an object
 in : x in y
 not in : x not in y
```

9 - Bitwise Operators

```
 & : AND : Copies 1 if both bits are 1, else use 0
 | : OR : Copies 1 if either of the bits is 1, else use 0
 ^ : XOR : Copies 1 if only one of the bits is 1, else use 0
 ~ : NOT : Simply flip the bits
 << : left shift : Shift left and fill by zero on right
 >> : right shift : Shift right and fill by zero on left
```

10 - Type Conversions

```
# Integers
a = int(121)    # 121
b = int(21.21) # 21
c = int("151") # 151
d = int("15f",16) # 351

# Floats
a = float("5.25e5") # 525000.0

# Booleans
a = ''
b = 'abc'
c = 12
d = bool(a) # False
e = bool(b) # True
f = bool(c) # True

# Strings
a = str("hello") # hello
b = str(51) # 51
```

```
c = str(25.25)  # 25.25
```

11 - Print & Input

```
### Input
## Read input as string from console
# Convert the input string accordingly
fruits = ["apple", "banana", "orange", "grapes"]
choice = input("Enter from 1 to 4:")

choice = int( choice )

if choice < 1 or choice > 4 :
        print( "Wrong choice" )
else :
        print( fruits[choice - 1] )

## Output
Enter from 1 to 4:3
orange

## Read multiple arguments
a, b, c = input( "Enter a, b, and c with space as delimiter:" ).split()
print( f"a: {a}, b: {b}, c: {c}")

## Multiple arguments output
Enter a, b, and c with space as delimiter:1 2 3
a: 1, b: 2, c: 3

### Print

# print literal values
print( "Hello Python !!" )
print( 15 )
print( 25.25 )

# print variables
name = "Ricky"
print( name )

# print expressions
start = 20
end = 50
print( end - start )
```

12 - Formatting

```
## Conversion Types
d, i - Signed integer decimal, i is not supported by format method
o - Unsigned octal, u - Unsigned decimal
x, X - Unsigned hexadecimal, leading 0x or 0X
e, E - Floating point exponential format
f, F - Floating point decimal format
c - Single character
r - String (converts python object using repr())
s - String (converts python object using  str())

name = "Joe"
age = 25
weight = 25.2456

# String and Number
print( f"{name}, {age}" ) # Joe, 25
print( "%(name)s, %(age)d" % {'name': name, "age": age} ) # Joe, 25
print( '{0:s}, {1:d}' . format( name, age ) ) # Joe, 25

# Fixed digit numbers
print( f"{name}, {age:03d}" ) # Joe, 025
print( "%(name)s, %(age)03d" % {'name': name, "age": age} ) # Joe, 025
print( '{0:s}, {1:03d}' . format( name, age ) ) # Joe, 025

# Floating point
print( f"{name}, {weight:3.2f}" ) # Joe, 25.25
print( "%(name)s, %(weight)3.2f" % {'name': name, "weight": weight} ) # Joe,
25.25
print( '{0:s}, {1:3.2f}' . format( name, weight ) ) # Joe, 25.25
```

13 - Sequences (Ordered Sets)

```
# string
# ordered, indexed, unchangeable (immutable), allows duplicates
hello = "Hello Python !!"
print( hello[0] ) # H
print( hello[4] ) # 0

print( hello[1,4] ) # syntax error, range not allowed
hello[0] = "K" # syntax error, immutable, item assignment not allowed

# list
# ordered, indexed, changeable (mutable), allows duplicates
fruits = ["apple", "banana", "orange", "grapes"]
```

```
print( fruits[0] ) # apple
print( fruits[2] ) # orange
print( fruits[1:3] ) # ['banana', 'orange']
print( fruits[:2] ) # ['apple', 'banana']
print( fruits[-3:-1] ) # ['banana', 'orange']
print( fruits[-3:] ) # ['banana', 'orange', 'grapes']

fruits[0] = "banana"
print( fruits[0] ) # banana
print( fruits[1] ) # banana

# tuple
# ordered, indexed, unchangeable (immutable), allows duplicates
vehicles = ("car", "bus", "truck")
print( vehicles[0] ) # car
print( vehicles[1] ) # bus

vehicles[0] = "bicycle" # syntax error, immutable, item assignment not
allowed

# range
# generates an immutable sequence of numbers over time
a = range(5)
b = range(0,5)
c = range(0,10,2)

print( a[2] ) # 2
print( c[2] ) # 4
```

14 - Binary Sequences

```
# bytes
# ordered, indexed, unchangeable (immutable), allows duplicates
hello1 = b"Hello Python !!"
hello2 = bytes('Hello Python !!', 'utf8')
print( hello1[0] ) # 72
print( hello2[4] ) # 111
print( hello1 ) # b'Hello Python !!'
print( hello1.decode() ) # Hello Python !!

print( hello[1,4] ) # syntax error, range not allowed
hello[0] = "K" # syntax error, immutable, item assignment not allowed

# bytearray
hello1 = bytearray( b"Hello Python !!" )
hello2 = bytearray( "Hello Python !!", "utf8" )
```

```
hello3 = bytearray( [72, 101, 108, 108, 111] )
print( hello1 ) # bytearray(b'Hello Python !!')
print( hello2 ) # bytearray(b'Hello Python !!')
print( hello3 ) # bytearray(b'Hello')
print( hello3.decode() ) # Hello
```

15 - List Comprehension

```
## list comprehension
# [ expression for item in list if conditional ]
# expression can be a valid expression including function call
# using if is optional

## Example 1 - Iterate the list and conditionally select item
fruits = ['watermelon', 'kiwi', 'litchi', 'guava', 'grapes', 'banana',
'apple']

filtered = [ fruit for fruit in fruits if fruit[0] == 'g' ]

print( filtered ) # ['guava', 'grapes']

## Example 2 - Call function
def table(num, x) :
    return num * x

num = 2
tableOfNum = [ table(num, x) for x in range(11) if x > 0 ]

print( tableOfNum ) # [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]

## Example 3
num = 2
tableOfNum = [ num * x for x in range(11) if x > 0 ]

print( tableOfNum ) # [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

16 - Mappings & Sets

```
# dict
# unordered, indexed, changeable (mutable), allows duplicates
person = { "name": "Joe", "age": 24 }
print( person["name"] ) # Joe
print( person["age"] ) # 24
print( person.get("name") ) # Joe
print( person ) # {'name': 'Joe', 'age': 24}

person["name"] = "Jack"
print( person["name"] ) # Jack

# convert list of list or tuple to dict
person1 = dict( [ [ "name", "Joe" ], [ "age", 24 ] ] )
person2 = dict( [ ( "name", "Joe" ), ( "age", 24 ) ] )
print( person1 ) # {'name': 'Joe', 'age': 24}
print( person2 ) # {'name': 'Joe', 'age': 24}

# set
# unordered, unindexed, changeable (mutable), no duplicates
fruits = {"apple", "banana", "orange", "grapes"}
print( fruits ) # {'apple', 'grapes', 'orange', 'banana'}
print( fruits ) # {'apple', 'grapes', 'banana', 'orange'}

a = fruits[0] # syntax error, unsubscriptable

# convert list or tuple to set
fruits1 = set( ["apple", "banana", "orange", "grapes"] )
fruits2 = set( ("apple", "banana", "orange", "grapes") )

print(fruits1) # {'apple', 'orange', 'grapes', 'banana'}
print(fruits2) # {'apple', 'orange', 'grapes', 'banana'}

# frozenset
# unordered, unindexed, unchangeable (immutable), no duplicates
```

17 - Conditional Statements

```
# Statement Blocks - if, else and elif
# Statement blocks works using indentations
# Statement block executed only if given condition is true

# if
fruits = ["apple", "banana", "orange", "grapes"]

if "banana" in fruits :
        print( "Found Banana" )

# else
if "banana" in fruits :
        print( "Found Banana" )
else :
        print( "Banana is missing" )
```

```
# elif
name = "Joe"

if name == "Rock" :
        print( "Rock is performing now" )
elif name == "Joe" :
        print( "Joe is performing now" )
else :
        print( "Stage is free" )
```

18 - Iterative Loop Statement

```
# Iterative Loop - for
# Statement blocks works using indentations
# Statement block executed for each item of container

# Iterate list
fruits = ["apple", "banana", "orange", "grapes"]

# Standard
for fruit in fruits :
    print( fruit )

# Iterate with index using range
for i in range(len(fruits)):
    print( "Fruit {}: {}" . format( i + 1, fruits[i] ) )

# Iterate with index using enumerate
for num, name in enumerate(fruits, start=1):
    print( "Fruit {}: {}" . format( num, name ) )

# Iterate set
fruits = {"apple", "banana", "orange", "grapes"}

for fruit in fruits :
    print( fruit )

# Iterate dict
person = { "name": "Joe", "age": 24 }

for key in person:
    print( "Person {}: {}" . format( key, person[key] ) )
```

19 - Conditional Loop Statement

```
# Conditional Loop - while
# Statement blocks works using indentations
# Statement block executed for each item of container till the condition is
true

# while with condition
limit = 5
start = 1
factor = 2
while start <= limit:
        result = start * factor
        print( "Multiplied {} by {}: {}" . format( start, factor, result ) )
        start = start + 1

# while with condition and break
fruits  = ["apple", "banana", "orange", "grapes"]
count   = len( fruits )
index   = 0

while index < count :
    if fruits[index] == "orange" :
        print( "Found orange" )
        break
    else :
                index = index + 1
                continue
```

20 - Functions

```
# Declaration & Definition
# def functionName(argument1, argument2, argument3)
# Declared using the def keyword
# Function name must be a valid identifier
# Function can have 0 to n arguments
# Function arguments can be literal values, variables (valid identifiers),
and expressions
# Function is a block of code which runs when it is called
# Function block can have multiple statements
# Function variables exists within the function blocks during the call

def sum(a, b) :
        return a + b

def subtract(a, b) :
        return a - b

def getPerson(name, age) :
        person = { "name": name, "age": age }
        return person

# Call
# Functions can be called by passing the arguments according to the
declaration

a = 20
b = 50
c = sum(a, b)
d = sum(b, 50)
e = subtract(b, a)
p = getPerson("Joe", 25)

print( "Sum - {} plus {}: {}" . format( a, b, c ) ) # Sum - 20 plus 50: 70
print( "Sum - {} plus 50: {}" . format( b, d ) ) # Sum - 50 plus 50: 100
print( "Subtraction - {} minus {}: {}" . format( b, a, e ) ) # Subtraction -
50 minus 20: 30
print( "Person - {}" . format( p ) ) # Person - {'name': 'Joe', 'age': 75}
```

21 - Generic Operations on Containers

```
# Containers: str, list, tuple, range, dict, set, frozenset, bytes, bytearray
# Uses keys for dictionaries

# Generic
# Items count: len(container) -> Returns items count
# Min value: min(container) -> Returns item having minimum value
# Max value: max(container) -> Returns item having maximum value
# Sorting: sorted(container) -> Returns items in sorted order
# Searching: in operator -> item in container, item not in container
# Iterate: enumerate(container) -> returns iterator to iterate the container
# All true: all(container) -> Returns True if all items evaluates to true
# Any one true: any(container) -> Returns True if any one of the items
evaluates to true

# Numeric
# Items sum: sum(container) -> Returns sum of all items value, works for
containers having numeric items

# Ordered Sequences
# Reversed: reversed(container) -> Returns the items in reversed order
# Item index: container.index(val) -> Finds the index of given item value
```

22 - String Operations

```
hello   = "Hello Python !!"
fruits  = ["apple", "banana", "orange", "grapes"]

# Split, Join, and Strip - split, join, and strip
print( hello.split( ' ' ) ) # Split using space as delimiter -> ['Hello',
'Python', '!!']
print( ',' . join( fruits )  ) # Join using comma as delimiter ->
apple,banana,orange,grapes
print( hello.strip() ) # Strips whitespace on left and right -> 'Hello Python
!!'

# Searching - startsWith and endsWith
# Tests whether sub-string is found
print( hello.startswith( "He" ) ) # True -> Without start and end parameters
print( hello.startswith( "llo", 2, 8 ) ) # True -> With start and end
parameters
print( hello.startswith( "Pyt", 6 ) ) # True -> With start parameter

# Searching - find and index
# Returns starting index if found, find returns -1, index throws exception
print( hello.find( "llo" ) ) # 2 -> Without start and end parameters
print( hello.find( "Pyth", 4, 11 ) ) # 6 -> With start and end parameters
print( hello.index( "llo" ) ) # 2 -> Without start and end parameters
print( hello.index( "Pyth", 4, 11 ) ) # 6 -> With start and end parameters

# Searching - count
print( hello.count( "Pyth" ) ) # 1 -> Found 1 occurance of Pyth
print( hello.count( "Pyth", 4, 11 ) ) # 1 ->Found 1 occurance of Pyth

# Case operations - upper, lower, title, swapcase, casefold, and capitalize
print( hello.upper() ) # all uppercase -> HELLO PYTHON !!
print( hello.lower() ) # all lowercase -> hello python !!
print( hello.title() ) # first letter capital, all words -> Hello Python !!
print( hello.swapcase() ) # swap all case -> hELLO pYTHON !!
print( hello.casefold() ) # aggressive version of lower, also converts unique
unicode characters -> hello python !!
print( hello.capitalize() ) # first uppercase, all lowercase -> # Hello
python !!

# Alignment - center, ljust, and rjust
print( hello.center( 50 ) )          #                  Hello Python !!
print( hello.center( 50, '#' ) )     # #################Hello Python
```

```
!!##################
print( hello.ljust( 50 ) )          # Hello Python !!
print( hello.rjust( 50 ) )          #
Hello Python !!

# partition, encode, zfill
# Partition: partition the string, return tuple having before, argument, and
after
print( hello.partition( 'lo ' ) ) # ('Hel', 'lo ', 'Python !!')
# Encoding: encodes using given encoding ->
print( hello.encode( encoding='UTF-8',errors='strict' ) ) # b'Hello Python
!!'
# Zero fill: fills the left spaces with zero
print( hello.zfill( 50 ) ) # 00000000000000000000000000000000000Hello Python
!!

# Formatting
person = "Person Name: {} Age: {} Profession: {}" . format( "Joe", 25,
"Accountant" )
print( person )
```

23 - List Operations

```
fruits = ["apple", "banana", "orange", "grapes"]

# append, extend, insert, and remove
fruits.append( "guava" ) # Appends the item at the end
print( fruits ) # ['apple', 'banana', 'orange', 'grapes', 'guava']
fruits.append( "guava" ) # Appends the item at the end
print( fruits ) # ['apple', 'banana', 'orange', 'grapes', 'guava', 'guava']
fruits.extend( [ "litchi", "watermelon" ] ) # Adds multiple items to the set
print( fruits ) # ['apple', 'banana', 'orange', 'grapes', 'guava', 'guava',
'litchi', 'watermelon']
fruits.insert( 3, "guava" ) # Adds the item at the given index
print( fruits ) # ['apple', 'banana', 'orange', 'guava', 'grapes', 'guava',
'guava', 'litchi', 'watermelon']
fruits.remove( "guava" ) # Removes first matching item
print( fruits ) # ['apple', 'banana', 'orange', 'grapes', 'guava', 'guava',
'litchi', 'watermelon']

# pop, copy, and clear
poppedItem = fruits.pop( 4 ) # Removes the item at the given index
print( poppedItem ) # guava
print( fruits ) # ['apple', 'banana', 'orange', 'grapes', 'guava', 'litchi',
'watermelon']
fruitsCopy = fruits.copy() # Shallow copy
```

```
print( fruitsCopy ) # ['apple', 'banana', 'orange', 'grapes', 'guava',
'litchi', 'watermelon']
fruitsCopy.clear()
print( fruitsCopy ) # []

# sort and reverse
fruits.sort()
print( fruits ) # ['apple', 'banana', 'grapes', 'guava', 'litchi', 'orange',
'watermelon']
fruits.reverse()
print( fruits ) # ['watermelon', 'orange', 'litchi', 'guava', 'grapes',
'banana', 'apple']

# read, update, delete
print( fruits[1] ) # orange
fruits[ 1 ] = "kiwi"
print( fruits ) # ['watermelon', 'kiwi', 'litchi', 'guava', 'grapes',
'banana', 'apple']
del fruits[ 1 ]
print( fruits ) # ['watermelon', 'litchi', 'guava', 'grapes', 'banana',
'apple']
```

24 - Dictionary Operations

```
person = { "name": "Joe", "age": 24, "dob": "25-11-2000", "profession":
"Software Developer" }

# keys, values, and items
# returns iterable for iterations
print( person.keys() ) # dict_keys(['name', 'age', 'dob', 'profession'])
print( person.values() ) # dict_values(['Joe', 24, '25-11-2000', 'Software
Developer'])
print( person.items() ) # dict_items([('name', 'Joe'), ('age', 24), ('dob',
'25-11-2000'), ('profession', 'Software Developer')])

# pop, popitem, copy, and clear
poppedItem = person.pop( "dob" ) # Removes the item having given key
print( poppedItem ) # 25-11-2000
print( person ) # {'name': 'Joe', 'age': 24, 'profession': 'Software
Developer'}
poppedItem = person.popitem() # Removes the item inserted last
print( poppedItem ) # ('profession', 'Software Developer')
print( person ) # {'name': 'Joe', 'age': 24}
personCopy = person.copy() # Shallow copy
print( personCopy ) # {'name': 'Joe', 'age': 24}
```

```
# add, read, get, and update
person[ "profession" ] = "Teacher"
print( person ) # {'name': 'Joe', 'age': 24, 'profession': 'Teacher'}
person[ "profession" ] = "Musician"
print( person ) # {'name': 'Joe', 'age': 24, 'profession': 'Musician'}
print( person[ "age" ] ) # 24
print( person.get( "age" ) ) # 24
person.update( { "age": 28 } )
print( person.get( "age" ) ) # 28

# delete
del person[ "age" ]
print( person ) # {'name': 'Joe', 'profession': 'Musician'}
```

25 - Set Operations

```
fruits = {"apple", "banana", "orange", "grapes"}

# add, update, remove, and discard
fruits.add( "guava" ) # Adds the item to the set if not exist
print( fruits ) # {'apple', 'banana', 'orange', 'guava', 'grapes'}
fruits.add( "guava" ) # Adds the item to the set if not exist
print( fruits ) # {'apple', 'banana', 'orange', 'guava', 'grapes'}
fruits.update( [ "litchi", "watermelon" ] ) # Adds multiple items to the set
print( fruits ) # {'apple', 'banana', 'litchi', 'orange', 'watermelon',
'guava', 'grapes'}
fruits.remove( "litchi" ) # Find and remove the item, throws error if not
found
print( fruits ) # {'apple', 'banana', 'orange', 'watermelon', 'guava',
'grapes'}
fruits.discard( "litchi" ) # Find and remove the item, does not throw any
error
print( fruits ) # {'apple', 'banana', 'orange', 'watermelon', 'guava',
'grapes'}

# pop, copy, and clear
poppedItem = fruits.pop() # Removes an arbitrary item
print( poppedItem ) # apple
print( fruits ) # {'banana', 'orange', 'watermelon', 'guava', 'grapes'}
fruitsCopy = fruits.copy() # Shallow copy
print( fruitsCopy ) # {'banana', 'guava', 'grapes', 'orange', 'watermelon'}
fruitsCopy.clear()
print( fruitsCopy ) # set()
```

26 - Lambda

```
# Lambda: Anonymous function
# Arguments: Accepts n arguments
# Expressions: Only one expression is allowed
# Syntax: lambda arguments : expression

addTo100 = lambda a : a + 100
print( addTo100( 50 ) ) # 150

multiplyBy2 = lambda a : a * 2
print( multiplyBy2( 25 ) ) # 50
```

27 - Classes & Objects

```
# Python is an object oriented programming language
# Almost everything in Python is an object
# Almost everything has attributes and methods
# All functions have a built-in attribute __doc__
# Some objects have neither attributes nor methods
# Some objects are final

class Person:
        def __init__(self, name, age, profession) :
                self.name = name
                self.age = age
                self.profession = profession

        def printInfo(self) :
                print( "Person Name: {} Age: {} Profession: {}" . format(
self.name, self.age, self.profession ) )

joe = Person("Joe", 35, "Software Engineer")

joe.printInfo() # Person Name: Joe Age: 35 Profession: Software Engineer
```

28 Modules & Packages

```
# Module: A file having Python code including statements and definitions is
called as Module.
# A module can define variables, functions, and classes.
# A module can also include code which can be executed.
# The file name must be module name appended by .py extension.
# Module name can be accessed within it using the global __name__.
# Package: A Python package is a collection of Python modules.
# Package is a directory having modules.
# Regular packages must have an additional __init__.py file, to distinguish
it from a directory.
# Namespace packages since Python 3.3 do not need __init__.py file

## Module - models.py
class Person:
        def __init__(self, name, age, profession):
                self.name = name
                self.age = age
                self.profession = profession

        def printInfo(self) :
                print( "Person Name: {} Age: {} Profession: {}" . format(
self.name, self.age, self.profession ) )

## Module - main.py
# Search for models.py in current directory, main.py directory
# Search for models.py in the list of directories in the PYTHONPATH
environment variable
import models # Use import mylib.models if models module is available in
mylib directory

joe = models.Person("Joe", 35, "Software Engineer")

joe.printInfo() # Person Name: Joe Age: 35 Profession: Software Engineer

## Module - main.py
from models import Person

joe = Person("Joe", 35, "Software Engineer")

joe.printInfo() # Person Name: Joe Age: 35 Profession: Software Engineer

## Execute main module on console
python main.py

## List directories where Python searches for the Module
import sys

print(sys.path)

## Sample Directory Structure
├── nspkg1
│   └── regpkg1
│       ├── __init__.py
│       └── vehicle.py
├── nspkg2
│   └── nspkg2sub1
│       └── person.py
├── main.py

## Imports in main.py
import nspkg1.regpkg1.vehicle
import nspkg2.nspkg2sub1.person
```

29 Errors & Exceptions

```
# Errors
# Program stops working on error
# Python raises exceptions when it encounter error
# Syntax or parsing errors are the most common for beginners

# Exceptions
# No syntax errors found, program starts execution
# Errors detected during execution are called exceptions
# Use try: except: finally: to catch and handle the exceptions
# Use try: except: finally: to avoid program termination on exceptions
# Use try: except: else: instead of try: except: finally: for alternate flows
# Multiple except can be use to catch the exceptions

## Program
a = 10 * (1/0)

## Throws division by zero exception and terminate the program
Traceback (most recent call last):
  File "", line 1, in
    a = 10 * (1/0)
ZeroDivisionError: division by zero

## Updated Program - Valid - Try: Except: Finally
b = 10
try:
    a = 10 * (1/b)
    print( "a = {}" .format( a ) )
except:
    print( "Caught divide by zero - while getting a" )
    print( "Execute on error - b must be non-zero value" )
finally:
    print( "Execute Always - normal and exceptional flow" )

## Output
```

```
a = 1.0
Execute Always - normal and exceptional flow

## Updated Program - Error - Try: Except: Finally
b = 0
try:
    a = 10 * (1/b)
    print( "a = {}" .format( a ) )
except:
    print( "Caught divide by zero - while getting a" )
    print( "Execute on error - b must be non-zero value" )
else:
    print( "Alternate to exceptional flow" )

## Output
Caught divide by zero - while getting a
Execute on error - b must be non-zero value
Execute Always - normal and exceptional flow
```

30 - Maths

```
# math.ceil( num ) - Smallest integer greater than or equal to num
# math.floor( num ) - Largest integer less than or equal to num
# math.fabs( num ) - Absolute of num
# math.factorial( num ) - Integer value of num factorial
# math.pow( num, power ) - num raised to the power
# math.sqrt( num ) - Square root of num
# math.isqrt( num ) - Integer square root of the nonnegative integer num
# math.asin( num ) - Arc sine of num radians
# math.acos( num ) - Arc cosine of num radians
# math.atan( num ) - Arc tangent of num radians
# math.sin( num ) - Sine of num radians
# math.cos( num ) - Cosine of num radians
# math.tan( num ) - Tangent of num radians
# math.dist( num1, num2 ) - Euclidean distance between two points num1 and
num2
# math.degrees( num ) - Angle num from radians to degrees
# math.radians( num ) - Degree num from degrees to radians
# math.exp(x) - e raised to the power x, e = 2.718281
# math.expm1(x) - e raised to the power x minus 1, e = 2.718281
# math.log( num[, base] ) - One argument -> natural logarithm of x (to base
e), Two arguments -> logarithm of x to the given base
# math.log2( num ) - base-2 logarithm of num

## Examples
import math
```

```
num = 12.25

print( math.ceil( num ) ) # 13
print( math.floor( num ) ) # 12
```

31 - Files

```
# Function: open
# Arguments: 1st -> File path with name, 2nd -> Mode, 3rd -> Encoding
# Modes: r -> read only, w -> write only, a -> append only, r+ -> read/write
# Optionally append t or b to the mode to specify among text or binary

## Example - Open
f = open( "test.txt" ) # same as open( "test.txt", "rt" )

## Example - Write
# Open the file test.txt in write mode
file = open( "test.txt", "w", encoding="utf8" )

file.write( "Hello Python !!\n" )
file.writelines( [ "Line 1\n", "Line 2\n", "Line 3\n" ] )
file.close()

## Example - With - Read all lines
with open( "test.txt", "r" ) as file: # Use file to refer to the file object

    file = open( "test.txt", "r" )

    text = file.read( 15 ) # Read first 15 characters

    print( text )

    file.seek( 0 )

    for line in file:
        print ( line.rstrip() )

## Example - try: except: finally: - Read all lines
try:
    file = open( "test.txt", "r" )

    text = file.read( 15 ) # Read first 15 characters

    print( text )
    file.seek( 0 )

    for line in file:
        print ( line.rstrip() )
except:
        print( "Failed to open the file" )
finally:
        file.close()

## Output
Hello Python !!
Hello Python !!
Line 1
Line 2
Line 3
```

The Python Cheatsheet by https://www.tutorials24x7.com. Download PDF.

Notes

This cheatsheet does not cover all the programming concepts of Python.
It can be considered as a reference to quickly revise or learn the basic concepts of Python.
Submit the Contact Form in case you face any issues in using it or finds any discrepancy.