Python Fundamental

Python

Introduction

Python is...

- Python is a widely used **high-level** programming language for **general-purpose** programming (...).
- An **interpreted** language, Python has a design philosophy that emphasizes **code readability** (...).
- The language provides constructs intended to enable writing clear programs on both a **small and large scale**.

Source: [Wikipedia](https://en.wikipedia.org/wiki/Python_(programming_language)

Website

https://www.python.org/

Basics

Hello World!

```
"""
Hello World!
"""

# My first program
print("Hello World!")

'''
Execute: $ python FILENAME.py
'''
```

Shebang

```
#!/usr/bin/python
"""
Shebang
- Indicates the interpreter to use
- See: https://en.wikipedia.org/wiki/Shebang_%28Unix%29
- Execute: $ .\_01_helloWorld.py
```

```
"""

# My first program
print("Hello World!")
```

Interactive Shell (REPL)*

```
$ python

$ print("Hello ... Terminal!")
$ a = 5
$ print(a * 2)

$ exit()
```

* REPL (READ-EVAL-PRINT-LOOP):

- **READ** whatever input we type in,
- **EVAL**uate it,
- **PRINT** the result and then
- **LOOP** back to the beginning.

Rules

```
"""
Rules
- Leading whitespaces (spaces and tabs) are used to
  define the level and the grouping of statements
- Case Sensitive
- Split code
- Naming: ^[a-zA-Z_$][a-zA-Z_$0-9]*$
"""

print("1")
print("2")

print("1"); print("2") # Possible, but bad practice

for i in range(2):
  print(i)

print()
# split code with \
print("Laborum reprehenderit voluptate dolore adipisicing \
dolore aliqua magna voluptate commodo reprehenderit sit.")

myString01 = ""
_myString01 = "" # can start with underscore

2Count # Error: cannot start with numbers

print("1") print("2") # Error: One statement per line

 print("1") # Error: starts with a space

for i in range(2):
  print(i)    # Error: Insconsistent identation
   print(i*2)

Print("Hello World!") # Error: Python is case sensitive!
```

Help

```
$ python                # open terminal
$ help()                # open interactive help
$ quit()                # exit interactive help

$ help(len)             # help(CommandName)

$ help(math)            # help(module)
$ help(math.factorial)  # help(function)

$ a = 5
$ help(a)               # help(object)
```

input

```
"""
input
"""

Name = input("Please enter your name:")
print("Hello,")
print(Name)
```

Comments

```
'''
Comments
'''

# single line comment

'''
Eiusmod pariatur in nisi est et anim. Enim laboris ullamco ullamco
nostrud ipsum consequat incididunt ea eu officia amet ipsum.
'''

"""
Eiusmod pariatur in nisi est et anim. Enim laboris ullamco ullamco
nostrud ipsum consequat incididunt ea eu officia amet ipsum.
"""

print("Done!!")
```

Data types

Scalars

Integers

```
"""
Integers
- Integers are only limited by available memory
"""

intValue = 10
print(intValue)
intValue = 0b10 # binary
print(intValue)
intValue = 0o10 # octal
print(intValue)
intValue = 0x10 # hexadecimal
print(intValue)

# Using constructors
print()
intValue = int(2.5)
print(intValue)
```

```
intValue = int(-2.5)
print(intValue)

# Using constructors with base
print()
intValue = int("10")
print(intValue)
intValue = int("10", 2)
print(intValue)
intValue = int("10", 8)
print(intValue)
intValue = int("10", 16)
print(intValue)
```

Float

```
"""
Float
"""

import sys

floatValue = 2.123
print(floatValue)

# Using constructors
print()
floatValue = float(5)
print(floatValue)
floatValue = float("5")
print(floatValue)

# scientific notation
print()
floatValue = 2e6
print(floatValue)
floatValue = 1.136e-25
print(floatValue)
print("{0:.28f}".format(floatValue))

# max. Value
print()
print("{0}".format(sys.float_info.max))
print("{0:f}".format(sys.float_info.max))
```

```
# Special values
print()
floatValue = float("nan") # Not A Number
print(floatValue)
floatValue = float("inf") # Infinity
print(floatValue)
floatValue = float("-inf") # Negative infinity
print(floatValue)

# Promotion
print()
floatValue = (2 + 1) * 3 + 1
print(floatValue)
floatValue = (2 + 1) * 3 + 1.0
print(floatValue)
```

None

```
"""
None
"""

NULL = None

print(NULL == None)
print(NULL is None)
```

bool

```
"""
bool
"""

falsey1 = False
truthy1 = True
print(falsey1, truthy1)

falsey1 = bool(0)
falsey2 = bool(0.0)
falsey3 = bool("")
falsey4 = bool([])
print(falsey1, falsey2, falsey3, falsey4)
```

```python
truthy1 = bool(1)
truthy2 = bool(0.1)
truthy3 = bool("a")
truthy4 = bool([1, 2])
truthy5 = bool("False")
truthy6 = bool(float("nan"))
print(truthy1, truthy2, truthy3, truthy4, truthy5, truthy6)
```

Strings

Strings

```python
"""
Strings
"""

# Using constructors
print()
print(str(100))
print(str(3.04))

print('Single quotes')
print("double quotes")

# print('Quotes must match.") # Error: quotes must match

print()
print('"Mixed" quotes')
print("'Mixed' quotes")

print()
# triple quotes - multiline string
message = '''Laborum reprehenderit voluptate dolore adipisicing
dolore aliqua magna voluptate commodo reprehenderit sit.'''
print(message)
message = """Laborum reprehenderit voluptate dolore adipisicing
dolore aliqua magna voluptate commodo reprehenderit sit."""
print(message)

print()
print(message[0])
```

Escape characters

```
'''
Escape characters
See: http://python-
reference.readthedocs.io/en/latest/docs/str/escapes.html
'''

print()
print('\'Escape\' quotes')
print('Escape \\ Backslash\n')
print("Laborum \treprehenderit \noluptate.")
```

Raw strings

```
'''
Raw strings
'''

myPath = r"c:\Windows\System32"
print(myPath)

url = r"""https://mail.google.com/mail/u/0/#inbox"""
print(url)
```

Concatenation

```
'''
Concatenation
'''

# Concatenation of adjacent literal strings
message = "Python" " is " "Great"
print(message)

print()
message = ("SELECT * "
          "FROM MyTable "
          "WHERE id > 0")
print(message)
message = "SELECT * " \
          "FROM MyTable " \
          "WHERE id > 0"
print(message)

print()
language = "Python"
message = language + " is great!"
print(message)

str1 = "SELECT * "
str2 = "FROM MyTable "
str3 = "WHERE id > 0"
message = str1 + \
          str2 + \
          str3
print(message)

print()
message = "{0}{1}{2}".format(str1, str2, str3)
print(message)
```

String format

```
"""
String format
See: https://pyformat.info/
"""

message1 = "Python"
message2 = "Great"
print("Message1:{}, message2: {}".format(message1, message2))
print("Message1:{0}, message2: {1}".format(message1, message2))

print()
print("{0} is {1}, {0} IS {1}!!!!".format(message1, message2))
print("{m1} is {m2}, {m1} IS {m2}!!!!".format(m2 = message2, \
                                              m1 = message1))

print()
print(message1)
print(message2)
print(message1, end="")
print(message2)
print(message1, end=" ")
print(message2)
print()
print("1/3: {0}".format(1/3))
print("1/3: {0:f}".format(1/3))
print("1/3: {0:.3f}".format(1/3))
print("1/3: {0:10.3f}".format(1/3))
print("1/3: {0:010.3f}".format(1/3))

print()
print("{0:20} is GREAT".format(message1))
print("{0:^20} is GREAT".format(message1))
print("{0:_^20} is GREAT".format(message1))

print("{0:>20} is GREAT".format(message1))
print("{0:_>20} is GREAT".format(message1))

print("{0:20} is GREAT".format(message1))
print("{0:_<20} is GREAT".format(message1))
```

Functions

```
"""
Functions
See: help(str)
"""

message1 = "Python"
message2 = "Great"
print(message1, "is", message2) # print(*args)
print(message1 + " is " + message2) # concatenation
print("{} is {}".format(message1, message2)) # format

message = message1 + " is " + message2
print(message.isnumeric())
print("123".isnumeric())

print()
print("to lower:", message.lower())
print("to upper:", message.upper())
print("swap case:", message.swapcase())
print("capitalize:", message.capitalize())
print("title:", message.title())

print()
message = (message1 + " is great").upper()
print(message)
print()
print("Count py:", str(message.count("PY")))
print("Count py:", str(message.count("Py")))
print("Count t:", str(message.count("T")))

print()
print("Find T:", str(message.find("T")))
print("Find Z:", str(message.find("Z")))
print("Z in:", str('Z' in message))

print()
print("Starts with P:", str(message.startswith("P")))
print("Ends with Z:", str(message.endswith("Z")))

print()
print("Replace:", message.replace("GREAT", "Amazing"))

print()
print(">".join(['A', 'B', 'C', 'D']))
print(" ".join(['1', '2', '3', '4']))

print()
print("Student1;10;11;12".split(';'))

print()
# Creates 3 partitions:
# before separator, separator and after separator
print("Team1 VS Team2".partition(" VS "))
print()
student = "Student1;10;11;12".partition(";")
print("name:{} grades:{}".format(student[0], student[2]))
```

Operators

Aritmetic Operators

```
'''
Aritmetic Operators
'''

a = 2 + 2
print("a = 2 + 2 = {}".format(a))
a = a - 2
print("a - 2 = {}".format(a))
```

```
a = a * 2
print("a = a * 2 = {}".format(a))
a = a / 2
print("a = a / 2 {}".format(a))

print()
a = a ** 2 # Power
print("a = a ** 2 = {}".format(a))

print()
# // Divide and round DOWN to the nearest whole number
b = a // 3
print("b = a // 3 {}".format(b))
b = -a // 3
print("b = -a // 3 = {}".format(b))

print()
b = (a * 3) // 2.5 # Modulo
print("b = (a * 3) // 2.5 = {}".format(b))
c = (a * 3) % 2.5 # Modulo
print("c = (a * 3) % 2.5 = {}".format(b))
c = (b * 2.5) + c
print("c = (b * 2.5) + c = {}".format(c))

print()
a += 2 # variable = variable operator value, a = a + 2
print("a += 2 (a={})".format(a))
a -= 2 #  a = a - 2
print("a += 2 (a={})".format(a))
a *= 2
print("a *= 2 (a={})".format(a))
a /= 2
print("a /= 2 (a={})".format(a))
a **= 2
print("a **= 2 (a={})".format(a))
a //= 2
print("a //= 2 (a={})".format(a))
a %= 2
print("a %= 2 (a={})".format(a))
```

Bitwise Operators

```
'''
Bitwise Operators
'''
```

```
a = 2 # "0010"
print("a = {0} - {0:04b}".format(a))

# left shift
print("a << 1 = {0} {0:04b}".format(a << 1)) #0100
print("a << 2 = {0} {0:04b}".format(a << 2)) #1000

# right shift
print("a >> 1 = {0} {0:04b}".format(a >> 1)) #0001
print("a >> 2 = {0} {0:04b}".format(a >> 2)) #0000

# and
print("5 & 3 = {0} {0:04b}".format(5 & 3)) # 0001
#   0101
#   0011
# = 0001

# or
print("5 | 3 = {0} {0:04b}".format(5 | 3)) # 0111
#   0101
#   0011
# = 0111

# xor
print("5 ^ 3 = {0} {0:04b}".format(5 ^ 3)) # 0110
#   0101
#   0011
# = 0110

# ~ invert same as: -(x+1)
print("~5 = {0} {1}".format(~5, bin(~5)))
# 0 0101 = 5
# 0 0110 = 5 + 1
# 1 0110 = - (5 + 1)
```

Logical Operators

```
'''
Logical Operators
'''

a = 10
b = 11
c = a

print("a < b = {}".format(a < b))
print("a > b = {}".format(a > b))
print("a <= b = {}".format(a <= b))
print("a >= c = {}".format(a <= c))

print()
print("a == c = {}".format(a == c))
print("not(a != c) = {}".format(not(a != c)))

print()
print("a != c = {}".format((a != c)))
print("not(a == c) = {}".format(not(a == c)))

print()
print("(a < b) and (c > b) = {}".format((a < b) and (c > b)))
# 1 1 1
# 1 0 0
# 0 1 0
# 0 0 0

print()
print("(a < b) or (c > b) = {}".format((a < b) or (c > b)))
# 1 1 1
# 1 0 1
# 0 1 1
# 0 0 0
```

in

```
'''
in
- Containers: str, list, dict, range, tuple, set, bytes
'''

item = "xa"
container = "Example"

print(item in container)
print(item not in container)

item = 12
container = [9, 10, 13]

print(item in container)
print(item not in container)
```

Control Flow

Decision

if

```
'''
if
'''

NUMBER = 17
inputValue = int(input("input a value: "))

if inputValue == NUMBER:
    print("Winner!!!")
print("Game over...")
```

if ... :

```
'''
if ... :
[else ... :]
'''

NUMBER = 17
inputValue = int(input("input a value: "))

if inputValue == NUMBER:
    print("Winner!!!")
else:
    print("Not a winner!")
print("Game over...")
```

if ... :

```
'''
if ... :
[elif ... :]
[else ... :]
```

```
'''

NUMBER = 17
inputValue = int(input("input a value: "))

if inputValue == NUMBER:
    print("Winner!!!")
elif inputValue > NUMBER:
    print("your number is greater")
else:
    print("your number is lower")
```

if, logical operators and nested if

```
'''
if, logical operators and nested if
'''

grade1 = 15
grade2 = 6

avgGrade = (grade1 + grade2) / 2.0

if avgGrade >= 9.5 and grade1 >= 9.5 and grade2 >= 9.5:
    print("Student is aproved")
    if avgGrade >= 17:
        print("Give him a little start!!")
elif avgGrade >= 7.5 and grade1 >= 7.5 and grade2 >= 7.5:
    print("Exam")
elif avgGrade >= 5.5 and (grade1 >= 15 or grade2 >= 15):
    print("Assigment")
else:
    print("Student is not aproved")
print("Done...")

print()
studentName = "Student 1"
isStudent = False
if studentName == "Student 1" or studentName == "Student 2" \
    or studentName == "Student 3":
    isStudent = True
print(isStudent)

isStudent = studentName in ["Student 1", "Student 2", "Student 3"]
print(isStudent)
```

chained

```
'''
chained
'''

grade1 = grade2 = grade3 = 15

print(grade1, grade2, grade3)

x = 0 <= grade1 <= 20 # same as 0 <= grade and 0 <= grade
print(x)

x = grade1 == grade2 == grade3 == 15
print(x)

i
```

Conditional expressions (ternary)

```
'''
Conditional expressions (ternary)
'''

grade = 9.5
value = "Approved" if grade >= 9.5 else "Not Approved"
print (value)

isValid = False
value = 1 if isValid else 0
print (value)

def One():
    print("First function")

def Two():
    print("Second function")

value = One if value else Two
value()
```

Repetition

While

```
'''
While
'''

NUMBER = 17
hit = False

while not hit:
    inputValue = int(input("input a value: "))
    if inputValue == NUMBER:
        print("Winner!!!")
        hit = True
    elif inputValue > NUMBER:
        print("your number is greater")
    else:
        print("your number is lower")
```

While

```
'''
While
'''

NUMBER = 17
hit = False

while not hit:
    inputValue = int(input("input a value: "))
    iif inputValue == NUMBER:
        hit = True
    elif inputValue > NUMBER:
        print("your number is greater")
    else:
        print("your number is lower")
else:
    print("Winner!!!")
```

For

```
'''
For
Iterables: str, list, dict, range, tuple, set, bytes
'''

iterable = "Example"

for item in iterable:
    print(item)
```

For

```
'''
For
'''

NUMBER = 17
hit = False

for attempt in range(5):
    if not hit:
        print("Attempt {}".format(attempt + 1))
        inputValue = int(input("input a value: "))
        if inputValue == NUMBER:
            print("Winner!!!")
            hit = True
        elif inputValue > NUMBER:
            print("your number is greater")
        else:
            print("your number is lower")
```

Break

```
'''
Break
'''

NUMBER = 17
```

```
for attempt in range(5): # 5 attempts
    print("Attempt {}".format(attempt + 1))
    inputValue = int(input("input a value: "))
    if inputValue == NUMBER:
        print("WINNER!!!!")
        break
    elif inputValue > NUMBER:
        print("your number is greater")
    else:
        print("your number is lower")
else: # only executes if for loop exits normally
    print("You didn´t make it")
```

Break

```
'''
Break
'''

while True:
    option = input("1. insert q or Q to exit: ")
    if option == 'Q' or option == 'q':
        break

while True:
    option = input("2. insert q or Q to exit: ")
    if option.upper() == 'Q':
        break

while True:
    if input("3. insert q or Q to exit: ").upper() == 'Q':
        break
```

Continue

```
'''
Continue
'''

for i in range(0, 50, 2):
    if i > 10 and i <= 20:
        continue
    print(i)
```

Functions

Funtions and Empty blocks (pass)

```
'''
Funtions and Empty blocks (pass)
'''

def SayHello():
    print("Hello")
    for i in range(4):
        pass
    print("there!")

SayHello()

def Test():
    '''
    I will implement this function later...
    '''
    pass

SayHello()
Test()
SayHello()
```

Parameters

```
'''
Parameters
'''

def WriteNames(value1, value2):
    print("{} : {}".format(value1, value2))

def Max(value1, value2):
    if value1 >= value2:
        print(value1)
    else:
        print(value2)

def Say(value, qty):
    print(value * qty)

WriteNames("Student1", "123")
WriteNames("Student2", "456")
WriteNames("Student3", "789")

val1 = 10
val2 = 12
print("Max ({}, {}): ".format(val1, val2))
Max(val1, val2)

Say("Python", 10)
```

Scope

```
'''
Scope
'''

Value = "Global Scope"

def WriteV1():
    Value = "Inner Scope"
    print(Value)

def WriteV2(Value): # by Value, overloading is not allowed
    print("Param. Value: " + Value)
    Value = "Inner Scope"
    print(Value)

def WriteV3():
    global Value
    print("Value: " + Value)
    Value = "Inner Scope"
    print(Value)

print(Value)
WriteV1()
print(Value)

print()
WriteV2(Value)
```

```
print(Value)

print()
WriteV3()
print(Value)
```

Default values and named parameters

```
'''
Default values and named parameters
'''

def Multiply(value, times=1):
    print(value * times)

def SumAndMultiply(value1, value2=0, times=1):
    val = value1 + value2
    Multiply(times=times, value=val)

Multiply("Python")
Multiply("Python", 3)
Multiply(3)
Multiply(3, 3)

print()
SumAndMultiply(10, 2)
SumAndMultiply(10, 1, 2)
SumAndMultiply(10, times=2)
SumAndMultiply(times=2, value2=2, value1=3)
```

Variable number of arguments

```
'''
Variable number of arguments
'''

def ConcatSpaced(*names):
    returnValue = ""
    for n in names:
        returnValue += n + " "
    print(returnValue)
```

```
def Sum(start, *numbers):
    for v in numbers:
        start += v
    print(start)

def Mean(**grades): # dictionary params
    mean = 0
    for name, grade in grades.items():
        ConcatSpaced(name, str(grade))
        mean += grade
    print("Mean: {:.2f}".format(mean/len(grades)))

ConcatSpaced("Python", "101")
ConcatSpaced("Python", "is", "really", "great!")

print()
Sum(0)
Sum(4, 3)
Sum(0, 3, 5, 4)

print()
Mean(Student1=20, Student2=2)
print()
Mean(Student1=20, Student2=2, Student3=15)
```

Return

```
'''
Return
'''

def Max(*numbers):
    max = None
    for value in numbers:
        if max is None or max < value:
            max = value
    return max

def IsGreater(value1, value2):
    if value1 > value2:
        return True
    else:
        return False

def IsGreaterV2(value1, value2):
    if value1 > value2:
        return True
    return False

print(Max(1, 17, 5, 2, 50, -75, 1))

print()
print(IsGreater(10, 5))
print(IsGreater(10, 11))

print()
print(IsGreaterV2(10, 5))
print(IsGreaterV2(10, 11))
```

Builtin Functions

del

```
'''
del
'''

A = "Student 1"
print(A)

del A       # Delete variable
print(A)    # Error: cannot access to A
```

dir and vars

```
'''
dir and vars
'''

import math

__version__ = 1.0   # attribute

def Test():
    pass

print(dir())        # list names from current module
print()
print(dir(math))    # list names from math module
print()
print(vars(math))   # list names and values

print()
var = "teste"
print(dir(var))

print()
var = 5
print(dir(var))
```

range

```
'''
range
'''

def PrintRange(values):
    print(", ".join([str(i) for i in values]))

# range (stop)
PrintRange(range(4))
PrintRange(range(10))

# range (start, finish, [step])
PrintRange(range(3, 20))
PrintRange(range(0, 20, 2))
PrintRange(range(1, 20, 2))
PrintRange(range(20, 0, -1))
PrintRange(range(-20, 0, 2))
PrintRange(range(0, -20, -2))
```

len

```
'''
len
- Sized: str, list, dict, range, tuple, set, bytes
'''

item = "Example"
```

```
print(len(item))

item = [10, 2, 25]
print(len(item))
```

type

```
'''
type
'''

print(type(None))
print(type(1))
print(type("Example"))
print(type([10, 11]))
```

enumerate

```
'''
enumerate
'''

items = ["Student1", "Student2", "Student3", "Student4"]

for counter, value in enumerate(items):
    print(counter, value)


print()
for counter, value in enumerate(items, start = 1):
    print(counter, value)

print()
enumeratedItem = list(enumerate(items, 1))
print(enumeratedItem)
print(enumeratedItem[1][1])
```

sum

```
'''
sum
'''

items = [10, 10, 5, 25, 25, 25.0]

print(sum(items))
```

any(iterable) and all(iterable)

```
'''
any(iterable) and all(iterable)
'''

v = all([True, True, True])
print(v)
v = all([True, False, False])
print(v)

v = any([True, False, False])
print(v)
v = any([False, False, False])
print(v)

print()
v = any([0, 0])
print(v)
v = any("")
print(v)
v = any("X")
print(v)

def isDiv3(value):
    return value % 3 == 0

print(isDiv3(9))
print(isDiv3(19))

print()
v = any(isDiv3(x) for x in range(1000,1051))
print(v)

v = all(name == name.title() for name in
['Student1','Student2','student3'])
```

```
print(v)
```

zip

```
'''
zip
'''

students = ["Student1", "Student2", "Student3", "Student4",
"Student5"]
grades1 = [12, 14, 15, 15]
grades2 = [13, 14, 14, 14, 10]

for item in zip(students, grades1, grades2):
    print(item)

for name, first, second in zip(students, grades1, grades2):
    print("{}: {}".format(name, (first+second) / 2))
```

ord

```
'''
ord
'''

def HashMe(text):
    hash_value = 0
    for e in text:
        hash_value += ord(e)
    return hash_value

def PrintMyChars(text):
    for e in text:
        print(ord(e), end=" ")

print(HashMe("Python is Great!"))
print(HashMe("Python is GREAT!"))

PrintMyChars("Python is Great!")
print()
PrintMyChars("Python is GREAT!")
```

chr

```
'''
chr
'''

chrs = [80, 121, 116, 104, 111, 110, 32, 105, 115, 32, 71, 114, 101,
97, 116, 33]

text = "".join(chr(x) for x in chrs)
print(text)
```

filter

```
'''
filter
'''

values = list(range(10))

def Even(number):
    return number % 2 == 0

values = filter(Even, values)

print("".join(str(x) for x in values))
```

sum

```
'''
sum
'''

values = list(range(10))

print(sum(values))
```

Modules

Import

```
'''
Import
'''

import math

print(math.sqrt(25))

print(dir())
```

From ... import....

```
'''
From ... import....
'''

from math import sqrt

print(sqrt(25))

print(dir())
```

Module

```
'''
Module
'''

# Attributes
__version__ = 1.0
__myOwnAttribute__ = 7

def MySum(value1, value2):
    return value1 + value2
```

```
def Daddy():
    return "Running from: " + __name__

if __name__ == '__main__':
    print(Daddy())
    print(MySum(10, 20))
```

import module

```
'''
import module
'''

import _03_module

print(_03_module.MySum(1.25, 3.2))
print(_03_module.Daddy())
print(_03_module.__version__)
print(_03_module.__myOwnAttribute__)
```

import module

```
'''
import module
'''

from _03_module import MySum, __version__, __myOwnAttribute__

print(MySum(1.25, 3.2))

print(__version__)
print(__myOwnAttribute__)
```

import modules

```
'''
import modules
'''
```

```
from _03_module import *

print(MySum(1.25, 3.2))

# Error: import * does not import attributes
print(__version__)
```

Packages

```
'''
Packages
- Packages(folder) MUST contain an __init__.py inside
- __init__.py can be an empty file
'''

from Package1 import ModuleTest
# "as alias" otherwise name clash on ModuleTest
from Package2 import ModuleTest as Second

print(ModuleTest.MyMultiplier(15, 10))
print(Second.MyMultiplier(15, 10))
```

Packages

```
'''
Packages
'''

from Package1 import ModuleTest
import Package2.ModuleTest

print(ModuleTest.MyMultiplier(15, 10))
print(Package2.ModuleTest.MyMultiplier(15, 10)) # Full Path
```

Nested Packages

```
'''
Nested Packages
'''
```

```
from Package1 import ModuleTest
from Package1.Package1_1 \
    import ModuleTest as third # Nested packages

print(ModuleTest.MyMultiplier(15, 10))
print(third.MyMultiplier(15, 10))
```

This is a module DocString

```
'''
This is a module DocString
Execute (on folder):
$ python
$ import _10_docString
$ help (_10_docString)
$ _10_docString.MyFunction(1, 2)
'''

def MyFunction(value1, value2):
    """
    This is a reST style docstring.
    More style at:
    https://stackoverflow.com/questions/3898572/
    :param value1: this is a first param
    :param value2: this is a second param
    :returns: this is a description of what is returned
    :raises keyError: raises an exception
    """
    return value1 + value2

if __name__ == "__main__":
    print(MyFunction(1, 2))
    print(MyFunction.__doc__)
```

try: import

```
'''
try: import
'''
try:
    import cProfile as profiler
```

```
except:
    import profile as profiler

print (profiler.__all__)
```

Collections

Sequences

```
'''
Sequences
- Sequences: str, list, tuple, range, bytes
'''

sequence = "Example"
print(sequence[1])
print(sequence.index("m"))
print(sequence.count("xa"))
print("".join(reversed(sequence)))

sequence = [10, 11, 13]
print(sequence[1])
print(sequence.index(10))
print(sequence.count(11))
print(" ".join(str(i) for i in reversed(sequence)))
```

Slicing

```
'''
Slicing
'''

students = ['student1',
            'student2',
            'student3',
            'student4',
            'student5']

for i in range(5):
    print('Student at {}: {}'.format(i, students[i]))

print()
for i in range(-1, -(len(students) + 1), -1):
    print('Student at {}: {}'.format(i, students[i]))

print()
print('Last element:', students[-1])

print()
print('First student first character:', students[0][0])

print(students[ 0 : 2 ])
print(students[ 1 : len(students) - 1 ])
print(students[ 1 : -2 ])
print(students[ 3 : ]) # 3 to the end
print(students[ : 3 ]) # start to 3
print(students[ : -2 ]) # start to -2
print(students[ : ])

print()
print(students[ : 4 : 2]) # increment of 2
print(students[  : -1 : 2])
print(students[::2])
print(students[::-1])
print(students[::-2])

print()
BackupList = students # refers to the same object
print(students)
print(BackupList)
students[0] = "XXX"
print(students)
print(BackupList)

print()
BackupList = students[:]  # slicing creates a copy
print(students)
print(BackupList)
students[0] = "YYY"
print(students)
print(BackupList)
```

Mutability

```
'''
Mutability
- Python is strongly object-oriented in the sense that \
  everything is an object including numbers, strings \
  and functions
- Immutable objects: int, float, decimal, complex, bool, \
                     string, tuple, range, frozenset, bytes \
- Mutable objects: list, dict, set, bytearray, \
                   user-defined classes
'''

def MemAddress(obj):
    return hex(id(obj))

def PrintMemory(obj):
    print(MemAddress(obj), ":", obj)

print(id(MemAddress))

# integers are immutable
print()
integerEx = 5
PrintMemory(integerEx)
integerEx += 1
PrintMemory(integerEx)

print()
value2 = integerEx
print(integerEx is value2) # identity equality

# tuples are immutable
print()
tupleEx = (1, 2, 3)
PrintMemory(tupleEx)
tupleEx += (4, 5, 6)
PrintMemory(tupleEx)

# list are mutable
print()
listEx = []
PrintMemory(listEx)
listEx += [11, 22]
PrintMemory(listEx)
listEx.append(33)
PrintMemory(listEx)
listEx.remove(11)
PrintMemory(listEx)

print()
List1 = [1, 2, 3]
List2 = [1, 2, 3]
print(List1 == List2)
print(List1 is List2)

print()
print(type(listEx[0]))
PrintMemory(listEx[0])
PrintMemory(listEx)
listEx[0] = 123
PrintMemory(listEx[0])
PrintMemory(listEx)
```

List

```
'''
List
 '''

import math

def Avg(list):
    sum = 0
    for grade in list:
        sum += grade
    return sum / len(list)

def ListOnebyOne(list):
    for i in range(len(list)):
        print(list[i])

def ListOnebyOneV2(list):
    for item in list:
        print(item)

def AppendValue(list, value):
    list.append(value)

def Change(list):
    list.append(10)
    list = [1, 1, 1, 1]
    print(list)

grades = [10, 8, 4, 17]

listCons = list("Student Grades")
print(listCons)

print('Grades:', grades)
print("Avg:", Avg(grades))

print()
grades.append(15)
print('Updated grades:', grades)
print('Updated avg:', Avg(grades))

print()
AppendValue(grades, 12)
print('Updated grades:', grades)
Change(grades)
print('Updated grades:', grades)

print()
ListOnebyOne(grades)
ListOnebyOneV2(grades)

print()
print("First grade:", grades[0])
del grades[0]
print('First grade:', grades[0])

print()
grades.append(12.5)
grades.sort()
print('Sorted grades:', grades)
grades.sort(reverse=True)
print('Sorted grades:', grades)
grades.reverse()
print('Sorted grades:', grades)

print()
print('Sorted grades:', sorted(grades, reverse=False))
print('Sorted grades:', grades)

grades += [18, 9, 7]
grades.extend([18, 9, 7])
print('extended:', grades)

print()
# List Comprehension
positiveGrades = [math.ceil(i) for i in grades if i > 9.5]
print(positiveGrades)

print()
print(", ".join(str(x) for x in positiveGrades))

print()
print('index of 17:', positiveGrades.index(17))
print()
positiveGrades.append(17)
print(positiveGrades)
positiveGrades.remove(17) # remove first
print(positiveGrades)

print(positiveGrades.index(10)) # raises ValueError exception
positiveGrades.remove(10) # raises ValueError exception
```

Tuple

```
'''
Tuple
'''

def Sum(values):
    sum = 0
    for item in values:
        if type(item) == int or type(item) == float:
            sum += item
        else:
            sum += Sum(item)
    return sum

def Len(values):
    sum = 0
    for item in values:
        if type(item) == int or type(item) == float:
            sum += 1
        else:
            sum += Len(item)
    return sum

if __name__ == "__main__":

    temp = tuple("Constructor")
    print(temp)
    temp = tuple(["Anoter", "Constructor", 2])
    print(temp)

    firstYearGrades = (15, 18, 12, 10)
    secondYearGrades = (13, 13, 12, 11)

    emptyTuple = ()
    singletonTuple = (2, ) # (2) is integer 2
    grades = (15, 20, firstYearGrades, secondYearGrades)
    # could be without parentheses, but bad practice
    # grades = 15, 20, firstYearGrades, secondYearGrades

    print()
    print(",".join(str(x) for x in firstYearGrades))

    del firstYearGrades
    del secondYearGrades

    temp = (10, ) * 3 # (10, 10, 10)
    grades += temp    # concatenation

    print()
    print(grades)
    print(len(grades))

    print()
    print(19 in grades)

    print()
    print("Sum:", Sum(grades))
    print("Grades:", Len(grades))
    print("AVG:", Sum(grades)/ Len(grades))

    print()
    print("First year grades: ", grades[2])
    print("Second year grades: ", grades[3])

    print()
    print("First year First grade: ", grades[2][0])
    print("First year Second grade: ", grades[2][1])
    print("First year Third grades: ", grades[2][2])
```

Tuple

```
'''
Tuple
'''

import _05_tuple1 as t

def Report(values):
    report = [] # empty list
    report.append(("sum", t.Sum(values)))
    report.append(("len", t.Len(values)))
    report.append(("Avg", t.Sum(values) / t.Len(values)))
    return report

firstYearGrades = (15, 18, 12, 10)
secondYearGrades = (13, 13, 12, 11)

grades = firstYearGrades, secondYearGrades

del firstYearGrades
del secondYearGrades

studentReport = Report(grades)
print(studentReport)

for item in studentReport:
    # Unpack same as desc, value = item[0], item[1]
    desc, value = item
    print(desc, ":", value)

# Unpacking
x, y = (5, 6)
print(x, y)
x, y = 5, 6
print(x, y)

x, y = y, x # Swap
print(x, y)

# * represents: rest of
values = list(range(10))
(one, two, *three) = values
print(three)
(one, *two, three) = values
print (two)
(*one, two, three) = values
print(one)
```

Dictionary

```
'''
Dictionary
- key:value pair.
- keys must be immutable and are unique.
'''

student = {}
```

```
student["john"] = "student1@email.com"
student["jack"] = "student2@email.com"

print("Number of students:", len(student))

print()
if "john" in student:
    print("John exit!!")
if "cliff" in student:
    print("cliff exit!!")

print()
# alias
backupStudent = student
student["john"] = "john@python.org"
print(student)
print(backupStudent)

print()
# copy() perform a shallow copy
backupStudent = student.copy()
backupStudent["john"] = "otherJohn@python.org"
print(student)
print(backupStudent)

print()
for name, email in student.items():
    print("Name: {} | email: {}".format(name, email))

print()
for name, email in sorted(student.items()):
    print("Name: {} | email: {}".format(name, email))
```

Dictionary

```
'''
Dictionary
'''

grades = {
    "student1" : [10, 11, 12],
    "student2" : [15, 14, 14],
    "student3" : [10, 10, 12],
    "student4" : [14, 14, 10]
}
```

```python
grades["student5"] = [10, 10, 10]
print(grades) # The order is defined by python.

del grades["student3"]
print(grades)

print()
print(grades.values())

print()
print("Number of students:", len(grades))

print()
if "student1" in grades:
    grades["student1"][0] = 13
    print(grades["student1"])
    grades["student1"].append(15)
    print(grades["student1"])

print()
# alias
backupGrades = grades # alias
backupGrades["student1"][0] = 1
print(backupGrades)
print(grades)

print()
# after copy() the LISTs (complex object) of grades
# will refer to the same objects, so remain equals in both
backupGrades2 = grades.copy()
backupGrades2["student5"][0] = 3
print(backupGrades2)
print(grades)

print()
for name, studentGrades in grades.items():
    print("Name: {} | Grades: {}".format(name, studentGrades))

for name, studentGrades in sorted(grades.items()):
    print("Name: {} | Grades: {}".format(name, studentGrades))

print()
print(",".join(str(x) for x in grades.items()))
for name, studentGrades in sorted(grades.items()):
    print(name + "," + ",".join(str(x) for x in studentGrades))

print()
for value in grades.values():
    print(value)
```

```
for keys in grades.keys():
    print(keys)

print()
Positive = { name: grade for name, grade
                         in grades.items()
                         if 10 in grade}
for name, studentGrades in Positive.items():
    print("Name: {} | Grades: {}".format(name, studentGrades))
```

Sets

```
'''
Sets
A set is an unordered collection with no duplicate elements.
Allow operations based on set theory.
'''

set1 = {"a", "b", "c", "a", "d", "e", "f"}
print(set1)
set2 = set("aaaabcd")
print(set2)

print()
print(set1 & set2)
print(set1.intersection(set2))

print()
print(set1 | set2)
print(set1.union(set2))

print()
print(set1 - set2)
print(set1.difference(set2))

print(set1.symmetric_difference( set("xzabc") )) # not in both

print()
print(set1.issuperset(set2))
print(set2.issubset(set1))
print(set2.isdisjoint(set("xz")))

print(set2.issuperset({"a", "b"}))
print(set2.issuperset({"a", "b"}))

print()
```

```
for item in set1 - set2:
    set2.add(item)
print(set2)

print()
set2.remove("a")
print(set2)

print()
print(",".join(str(x) for x in set2))

print()
print("b" in set2)

print()
words = ("stu", "stud", "student", "dent")
w = {x for x in words if len(x) == 4} # Set Comprehension
print(w)

print()
set2.clear()
print(set2)
```

Input and Output

Files

```
'''
Files
open(FILE, ACCESSMODE)
ACCESSMODE:
w, r, a, - write, read, append
w+, r+, a+ - write/read, read/write, append/read
wb, rb, ab - write, read append binary
wb+, rb+, ab+ - write/read, read/write, append/read binary
'''

import os
import time
from pathlib import Path

thisFile = open(os.path.realpath(__file__), "r")
strContent = thisFile.read()
print(strContent)
thisFile.close()
```

```
thisFile = open(os.path.realpath(__file__), "r")
done = False
while True:
    line = thisFile.readline()
    if len(line) == 0:
        break
    print(line)
    if not done:
        print("FIRST LINE AGAIN")
        thisFile.seek(0)
        done = True
thisFile.close()
del thisFile

filePath =
str(Path(os.path.dirname(__file__)).parent.joinpath('Temp',
'files.txt'))
fileIn = open(os.path.realpath(__file__), "r")
fileOut = open(filePath, 'w+')

fileOut.write("{}\n\n".format("_" * 40))
fileOut.write("Backup created: ")
fileOut.write(time.strftime("%Y-%m-%d %H:%M:%S") + "\n")
fileOut.write("{}\n\n".format("_" * 40))

for line in fileIn:
    print(line)
    fileOut.write(line)
fileIn.close()
fileOut.close()
```

pickle

```
'''
pickle
'''

import pickle
import os
from pathlib import Path

grades = {
    "student1" : [10, 11, 12],
    "student2" : [15, 14, 14],
    "student3" : [10, 10, 12],
    "student4" : [14, 14, 10]
}

filePath =
str(Path(os.path.dirname(__file__)).parent.joinpath('Temp',
'pickle.dat'))

file = open(filePath , 'wb' )
pickle.dump(grades, file)
file.close()

file = open(filePath, 'rb' )
grades2 = pickle.load(file)
print(grades2)
print(len(grades2))
print(grades2["student1"][1])
```

Exceptions

try ... except...

```
'''
try ... except...
'''

try:
    a = 1 / 2
except:
    print("Something went wrong")

try:
    a = 1 / 0
except:
    print("Something went wrong")
else: # Executes if no exceptions are raised
    print("Everithings Ok.")
```

try ... except...

```
'''
try ... except...
```

```
'''

try:
    a = 1 / 0
except ZeroDivisionError:
    print("Something went wrong with a zero value")
except ValueError:
    print("Something went wrong with values")
except:
    print("Something went wrong")
else: print("Everithings Ok.")

try:
    a = 1 / 0
except (ZeroDivisionError, ValueError):
    print("Something went wrong with values")
except:
    print("Something went wrong")
else: print("Everithings Ok.")
```

try ... except...

```
'''
try ... except...
'''

import os
from pathlib import Path

def process():
    filePath = os.path.join(str(Path( \
        os.path.dirname(__file__)).parent.joinpath('Temp')), "X")
    file = open(filePath)
    file.close()
    pass

try:
    process()
except OSError as error:
    print('Could not process. Error{}'.format(str(error)))

try:
    a = 1 / 0
except ZeroDivisionError as err:
    print("Error:", err)
except TypeError as err:
    print("Error:", err)
    print("Could not convert data.")
else:
    print('Everithings Ok.')
```

sys.exc_info

```
'''
sys.exc_info
'''

import sys

try:
    b = "1"
    a = 1 / b
except (TypeError) as err:
    print("Error:", sys.exc_info())
except:
    print("ERROR:", sys.exc_info()[0])
else:
    print('Everithings Ok.')
```

sys.exc_info

```
'''
sys.exc_info
'''

import sys

def handleError(err):
    if err[0] == TypeError:
        print("Type Error!")
    elif err[0] == ZeroDivisionError:
        print("Zero Division Error!")
    else:
        print("Ups!")

try:
    a = 1 / "0"
except:
    handleError(sys.exc_info())
else:
    print('Everithings Ok.')
```

finally

```
'''
finally
'''

try:
    b = 5
    if b == 5:
        raise ValueError("Cannot divide by five :)")
    a = 1 / b
    print(a)
except ValueError as err:
    print("My exceptionError: {0}".format(err))
except ZeroDivisionError as err:
    print("Error: {0}".format(err))
except TypeError:
    print("Could not convert data to an integer.")
else:
    print('Everithings Ok.')
finally:
    print('Clean things')
```

raise

```
'''
raise
'''

try:
    b = 5
    if b == 5:
        raise ValueError("Cannot divide by five :)")
    a = 1 / b
    print(a)
except ValueError as err:
    print("My exceptionError: {0}".format(err))
except ZeroDivisionError as err:
    print("Error: {0}".format(err))
except TypeError:
    print("Could not convert data to an integer.")
else:
    print('Everithings Ok.')
```

try ... except...

```
'''
try ... except...
'''

import sys

def A(option):
    B(option)

def B(option):
    C(option)
    if option == 3:
        raise ZeroDivisionError("UPS")

def C(option):
    D(option)
    if option == 2:
        raise TypeError("TypeError")

def D(option):
    try:
        E(option)
    except AttributeError:
        print("AttributeError Caught in D")

def E(option):
    if option == 1:
        raise AttributeError

try:
    A(3)
    print("Done!")
except TypeError as err:
    print(err, "Caught in main try...except block")
except:
    print(sys.exc_info()[0].__name__, "Caught on default except")

print("Finished...")
```

with

```
'''
with
- with EXPR as VAR: BLOCK
- call VAR.__enter__ method at the begining
- call VAR.__exit__ method at the end
'''

import os
import inspect
from pathlib import Path

filePath =
str(Path(os.path.dirname(__file__)).parent.joinpath('Temp',
'with.txt'))

# Guaranteed to close the file,
with open(filePath, 'w') as file:
    file.write('Hi there!')
    print(dir(file))

print()
a = [2, 4, 2]
print(dir(a))

class temp:
    def __enter__(self):
        print("__enter__")
        return self
    def Using(self):
        print("Using()")
    def __exit__(self, exc_type, exc_val, exc_tb):
        print("__exit__")

print()
print(dir(temp))

print()
with temp() as t:
    t.Using()
```

Standard Library

The Python Standard Library

https://docs.python.org/3/library/

Modules

sys

```
'''
sys
'''

import sys
from pprint import pprint

def sum(numbers):
    sum = 0
    for i in numbers:
        sum += int(i)
    return sum

print("Command line arguments:")
for i in sys.argv:
    print(i)

sys.argv.pop(0)
print("SUM: ", sum(sys.argv))

print(sys.getdefaultencoding())

print(sys.version_info)
print(sys.executable)

print(sys.platform)


print()
pprint(sys.path)

print()
pprint(vars(sys))

print("Done!")

sys.exit(0)
```

```
print("You do not see me!")
```

sys

```
'''
sys
'''

import sys

def Main():
    if len(sys.argv) < 2:
        sys.exit('Command line arguments are missing')
    arg = sys.argv[1]
    result = int(arg)
    #print(0 if result else 1)
    return 0 if result else 1

if __name__ == '__main__':
    sys.exit(Main())
```

csv

```
'''
csv
'''

import os
import csv
import random
from pathlib import Path

def WriteCSV(filepath, data, delim = ","):
    with open(filepath, "w", newline='') as file:
        writer = csv.writer(file, delimiter=delim)
        for line in data:
            writer.writerow(line)

def WriteCSV_DICT(filepath, data, headers, delim = ","):
    with open(filepath, "w", newline='') as file:
        writer = csv.DictWriter(file, delimiter=delim,
fieldnames=headers)
```

```
        writer.writeheader()
        for row in data:
            writer.writerow(row)

def ReadCSV(filepath, delim = ",") :
    with open(filepath, "r") as file:
        data = csv.reader(file, delimiter=delim)
        for row in data:
            print("\t".join(row))

if __name__ == "__main__":

    filePath =
str(Path(os.path.dirname(__file__)).parent.parent.joinpath('Temp',
'data.csv'))

    Students = [
        ["student 1", random.randint(0, 20), random.randint(0, 20),
random.randint(0, 20)],
        ["student 2", random.randint(0, 20), random.randint(0, 20),
random.randint(0, 20)],
        ["student 3", random.randint(0, 20), random.randint(0, 20),
random.randint(0, 20)],
        ["student 4", random.randint(0, 20), random.randint(0, 20),
random.randint(0, 20)]
    ]

    WriteCSV(filePath, Students)
    ReadCSV(filePath)

    print('-' * 50)
    print('-' * 50)

    # Creates dict
    header = ["Name", "Grade 1", "Grade 2", "Grade 3"]
    newStudents = [ dict(zip(header, row)) for row in Students]
    print( "\n".join([str(row) for row in newStudents]))

    print()
    WriteCSV_DICT(filePath, newStudents, header)
    ReadCSV(filePath)
```

Datetime

```
'''
```

```
Datetime
See: http://strftime.org/
'''

import datetime

OneDate = datetime.date(2017, 6, 10)
OneDateTime = datetime.datetime(2017, 6, 10, 11, 30, 0)

print(OneDate)
print(OneDateTime)
print(OneDateTime.year)
print(OneDateTime.month)
print(OneDateTime.day)
print(OneDateTime.hour)
print(OneDateTime.minute)
print(OneDateTime.second)

currentDate = datetime.date.today()
print(currentDate)

print(currentDate.strftime("%d/%m/%y")) # str from date
print(currentDate.strftime("%Y %b %d %a"))
print(currentDate.strftime("%Y %B %A"))

date = input("Insert date [%d/%m/%y]: ")
date = datetime.datetime.strptime(date, "%d/%m/%y") # date from
string
print(date.strftime("%Y %b %d %a"))

diff = currentDate - date.date()

print(diff)
print(diff.days)

now = datetime.datetime.now()
print(now.hour)
print(now.minute)
print(now.second)

print(now.strftime("%d/%m/%y %H:%M:%S %I%p"))
```

Time

```
'''
```

```
Time
See: https://docs.python.org/3.6/library/time.html
'''

import time

def SleepInc(value):
    for x in range(value):
        print("Sleeping for {} seconds".format(x))
        time.sleep(x)

print(time.strftime('%Y%m%d%H%M%S'))
print(time.strftime('%Y-%m-%d %H:%M:%S'))

print(time.ctime())

print(time.gmtime(0)) # Epoch

T = time.time() #ticks
print(T, "ticks")
T = time.localtime(T) #struct
print(T)
print(T[0])
T = time.asctime(T)
print(T)

SleepInc(5)
```

logging

```
'''
logging
'''

import os
import logging
from pathlib import Path

filePath =
str(Path(os.path.dirname(__file__)).parent.parent.joinpath('Temp',
'logging.log'))

print("Logging to", filePath)

logging.basicConfig( \
    level=logging.DEBUG, \
    format='%(asctime)s : %(levelname)s : %(message)s', \
    filename=filePath, \
    filemode='w') # a to append

logging.debug("Starting...")
for i in range(100, -1, -1):
    logging.info("iteration: {}".format(i))
    if i == 50:
        logging.warning("half the way")
    elif i == 25:
        logging.critical("A critical message at 25")
    elif i == 10:
        logging.error("An error message at 10")
logging.debug("Done...")
```

math

```
'''
math
'''

import math
from pprint import pprint

print(math.sqrt(25))
print(math.factorial(5))

pprint(vars(math))
```

os

```
'''
os
'''

import os
from pathlib import Path
import pprint as pp

def touch(filename):
    with open(filename, 'w') as file:
        file.write("")

print("__file__: ", __file__)
print("basename: ", os.path.basename(__file__))
print("dirname: ", os.path.dirname(__file__))
print("realpath: ", os.path.realpath(__file__))
print("relpath: ", os.path.relpath(__file__))
print("split: ", os.path.split(__file__))

tempfolder =
str(Path(os.path.dirname(__file__)).parent.parent.joinpath('Temp',
'FOLDER'))
tempfolder2 =
str(Path(os.path.dirname(__file__)).parent.parent.joinpath('Temp',
'FOLDER2'))

print(tempfolder, "\\n", tempfolder2)

if not os.path.exists(tempfolder):
    os.mkdir(tempfolder)

if not os.path.exists(tempfolder2):
    os.mkdir(tempfolder2)
    os.rmdir(tempfolder2)

print("isdir: ", os.path.isdir(__file__))
print("isfile: ", os.path.isfile(__file__))
print("join: ", os.path.join(tempfolder, "Temp"))

print()
file1 = os.path.join(tempfolder, "tempFile.tmp")
file2 = os.path.join(tempfolder, "back.txt")
touch(file1)
os.rename(file1, file2)
os.startfile(file2)
os.remove(file2)

tempfolder = str(Path(tempfolder).joinpath("1", "2", "3"))
os.makedirs(tempfolder)

print()
print(os.getcwd()) # Get current folder
os.chdir("c:\\")
print(os.getcwd()) # Get current folder

print()
print(os.name)
print(os.sep)
print()
print(os.environ["PROGRAMFILES"])
print(os.getenv("PROGRAMFILES"))
for i in os.environ.items():
    #print(i)
    pass

for root, dirs, files in
os.walk(str(Path(os.path.dirname(__file__)).parent)):
    print(root, dirs, files)
    print()
```

pathlib

```
'''
pathlib
See: https://docs.python.org/3.6/library/pathlib.html
'''

from pathlib import Path

path = Path(r'C:\Program Files').parent
print(path)

path = path.joinpath('Temp')
print(path)
```

Pretty print

```
'''
Pretty print
'''

import pprint

grades = {
    "student1" : [10, 11, 12],
    "student2" : [15, 14, 14],
    "student3" : [10, 10, 12],
    "student4" : [14, 14, 10]
}
```

```
print(grades)
print()
pprint.pprint(grades)
```

Urllib

```
'''
Urllib
'''

import urllib.request

with urllib.request.urlopen('http://python.org/') as response:
    html = response.read()
    # print(html) bytes
    print(html.decode('utf-8'))

with urllib.request.urlopen('http://www.google.com/') as everyWords:
    words = []
    for line in everyWords:
        # line_words = line.decode('utf-8').split()
        line_words = line.decode('latin1').split()
        for word in line_words:
            words.append(word)
    print(words)
```

this

```
"""
this
"""

import this
```

turtle

```
'''
turtle
```

```
'''

import turtle
import random

import _13_msvcrt_tty

MyTurtle = turtle.Turtle()

def circle(radius, color):
    MyTurtle.color('red')
    MyTurtle.begin_fill()
    MyTurtle.circle(radius)
    MyTurtle.end_fill()

for color in range(4):
    circle(random.randint(-30, 30), color)
    MyTurtle.forward(100)
    MyTurtle.right(90)

MyTurtle.hideturtle()

input() # Wait for user input to close window
```

msvcrt and tty

```
"""
msvcrt and tty
- GetKey() - Detect keypress
"""

try:
    import msvcrt # Windows
    def GetKey():
        return msvcrt.getch()
except ImportError:
    import sys
    import tty # Unix
    import termios
    def GetKey():
        fd = sys.stdin.fileno()
        original_att = termios.tcgetattr(fd)
        try:
            tty.setraw(sys.stdin.fileno())
            ch = sys.stdin.read(1)
        finally:
            termios.tcsetattr(fd, termios.TCSADRAIN, original_att)
        return ch

if __name__ == "__main__":
    print("Press a key to continue...")
    GetKey()
    print("Done.")
```

random

```
"""
random
"""
import random

start = 0
end = 100

print(random.randint(start, end))

print(random.random())

print(random.randrange(end))
```

configparser

```
"""
configparser
"""

import configparser
import os
from pathlib import Path

def SaveConfigFile(filepath, configobject):
    with open(filepath, "w") as file:
        configobject.write(file)

def CreateConfig(filepath):
    conf = configparser.ConfigParser()
    conf.add_section("Settings")
    conf.set("Settings", "font", "Consolas")
    conf.set("Settings", "font_size", "12")
    conf.set("Settings", "font_style", "Normal")
    conf.set("Settings", "font_info", \
        "%(font)s %(font_size)s pt %(font_style)s")
    SaveConfigFile(filepath, conf)

def GetConfig(filepath):
    conf = configparser.ConfigParser()
    conf.read(filepath)
    return conf

def GetSetting(filepath, section, setting):
    return GetConfig(filepath).get(section, setting)

def SetSetting(filepath, section, setting, value):
    conf = GetConfig(filePath)
    conf.set(section, setting, value)
    SaveConfigFile(filepath, conf)

def RemoveSetting(filepath, section, setting):
    conf = GetConfig(filepath)
    conf.remove_option(section, setting)
    SaveConfigFile(filepath, conf)

if __name__ == "__main__":

    filePath =
str(Path(os.path.dirname(__file__)).parent.parent.joinpath('Temp',
'settings.ini'))

    CreateConfig(filePath)

    SetSetting(filePath, "Settings", "font", "Helvetica")
    print(GetSetting(filePath, "Settings", "font_info"))

    SetSetting(filePath, "Settings", "XXXXX", "XXXXX")
    print(GetSetting(filePath, "Settings", "XXXXX"))

    RemoveSetting(filePath, "Settings", "XXXXX")
```

platform

```
'''
platform
```

```
'''

import os
import platform
import pprint as pp

def NewFileAtHome(fileName):
    if platform.platform().startswith('Windows'):
        return os.path.join(os.getenv('HOMEDRIVE'), \
            os.getenv('HOMEPATH'), fileName)
    return os.path.join(os.getenv('HOME'), fileName)

file = NewFileAtHome('test.log')
print("Logfile:", file)

pp.pprint(vars(platform))
```

smtp

```
'''
smtp
'''

import smtplib

EMAIL_HOST = 'smtp.mailtrap.io'
EMAIL_HOST_USER = '1ccd1fc0462e08'
EMAIL_HOST_PASSWORD = 'f2d750abd80bd4'
EMAIL_PORT = '2525'

from_email = "oscar.m.oliveira@gmail.com"

to = ["oscar.m.oliveira@gmail.com", "oscar.m.oliveira@outlook.com"]

subject = "Some subject..."
text = "Some text..."

body = "From: {}\r\nTo: {}\r\nSubject:
{}\r\n\r\n{}".format(from_email, ', '.join(to), subject, text)

server = smtplib.SMTP(EMAIL_HOST, EMAIL_PORT)
server.login(EMAIL_HOST_USER, EMAIL_HOST_PASSWORD)
server.sendmail(from_email, to, body)
server.quit()
```

```
print("Done!")
```

sqlite

```
'''
sqlite
'''
import sqlite3
import os
from pathlib import Path

conn = sqlite3.connect(':memory:') # Create a database in RAM
#conn = sqlite3.connect(filePath)

cursor = conn.cursor()
cursor.execute("""CREATE TABLE users(id INTEGER PRIMARY KEY, name 
TEXT,
    phone TEXT, email TEXT unique, password TEXT)""")

students = [ ["Student1", "123456789", "student1@email.com", 
"123456"],
    ["Student2", "987654321", "student2@email.com", "654321"]]


query = """INSERT INTO users(name, phone, email, password)
    VALUES ({})""".format(",".join(["'" + x + "'" for x in 
students[0]]))
cursor.execute(query)
#print(query, cursor.lastrowid)

query = """INSERT INTO users(name, phone, email, password)
    VALUES ({})""".format(",".join(["'" + x + "'" for x in 
students[1]]))
cursor.execute(query)
#print(query, cursor.lastrowid)

print()
cursor.execute('''SELECT * FROM users''')
for row in cursor.fetchall():
    print('{}: {}, {}, {}'.format(row[0], row[1], row[2], row[3]))

conn.commit()

id = 1
```

```
print()
query = """UPDATE users SET password= 'ABCDEF',
email='newemail@newemail.com'
    WHERE id = {}""".format(id)
cursor.execute(query)

cursor.execute("""SELECT * FROM users WHERE id={}""".format(id))
userInfo = cursor.fetchone()
print(userInfo)

print()
query = """DELETE FROM users WHERE id = 1"""
cursor.execute(query)

cursor.execute('''SELECT * FROM users''')
for row in cursor: # same as for row in cursor.fetchall():
    print('{}: {}, {}, {}'.format(row[0], row[1], row[2], row[3]))

conn.commit()

conn.close()

filePath = str(Path(os.path.dirname(__file__)).parent.parent. \
    joinpath('Temp', 'mydatabase.db'))
try:
    db = sqlite3.connect(filePath)
    cursor = db.cursor()
    cursor.execute("""CREATE TABLE IF NOT EXISTS users(id INTEGER
        PRIMARY KEY, name TEXT, phone TEXT, email TEXT unique,
        password TEXT)""")
    db.commit()
except Exception as e:
    db.rollback()
    raise e
finally:
    db.close()
```

subprocess

```
'''
subprocess
See: https://docs.python.org/3/library/subprocess.html
'''

import subprocess
```

```
program = "notepad.exe"
programargs = ["ping", "www.google.com"]

subprocess.call(program)

code = subprocess.call(programargs) # arguments
print(code)
if code == 0: print("Success!")

process = subprocess.Popen(program) # new process, dows not wait.

code = subprocess.Popen(program).wait()

subprocess.Popen([program, __file__])

process = subprocess.Popen(programargs, stdout=subprocess.PIPE)
data = process.communicate()
for line in data:
    print(line, "\n")
```

threading

```
'''
threading
'''

import random
import time
from threading import Thread

class MyThread(Thread):

    def __init__(self, name):
        Thread.__init__(self)
        self._name = name
        self._duration = random.randint(2, 20)

    def run(self):
        time.sleep(self._duration)
        msg = "Done: {} in {}".format(self._name, self._duration)
        print(msg)

def create_threads():
    for i in range(1, 6):
        name = "Thread n:{}".format(i)
        my_thread = MyThread(name)
        my_thread.start()

if __name__ == "__main__":
    create_threads()
```

operator

```
'''
operator
'''

import operator

def Calculate(op, max):
    print("\n")
    for x in range(1, max+1):
        print()
        for y in range(1, max+1):
            result = op(x, y)
            if isinstance(result, float):
                print("{:7.2f}".format(result), end="")
            else:
                print("{:5}".format(result), end="")

for o in (operator.add, operator.sub, operator.mul,
operator.itruediv):
    Calculate(o, 10)
```

xml - minidom

```
'''
xml - minidom
'''

import xml.dom.minidom
import urllib.request
import os
from pathlib import Path
from pprint import pprint as p
```

```
class StudentParser(object):

    def __init__(self, url):
        self.students = []
        self._enrolled = 0
        students = self.GetXML(url)
        self.GetStudents(students)

    def GetXML(self, url):
        try:
            f = urllib.request.urlopen(url)
        except urllib.error.URLError:
            f = url
        doc = xml.dom.minidom.parse(f)
        return doc.documentElement

    def GetText(self, node):
        return " ".join(t.nodeValue \
                        for t in node[0].childNodes \
                        if t.nodeType == t.TEXT_NODE)

    def GetGrades(self, node):
        grades = []
        for n in node.getElementsByTagName("grade"):
            number = n.getAttribute('number')
            grade = n.firstChild.nodeValue
            grades.append((number, grade))
        return grades

    def GetStudents(self, xml):
        self._enrolled = xml.getAttribute('enrolled')
        for node in xml.getElementsByTagName('student'):
            number = node.getAttribute('number')
            name = self.GetText(node.getElementsByTagName("name"))
            grades =
self.GetGrades(node.getElementsByTagName("grades")[0])
            self.students.append({"name": name, "number": number, \
                "grades": grades})

if __name__ == "__main__":
    filePath =
str(Path(os.path.dirname(__file__)).parent.parent.joinpath('Temp',
'students.xml'))
    print(filePath)
    students = StudentParser(filePath)
    p(students.students)
```

xml - etree.ElementTree

```
'''
xml - etree.ElementTree
'''

try:
    # faster C implementation
    import xml.etree.cElementTree as ET
except ImportError:
    import xml.etree.ElementTree as ET
import os
import datetime
from pathlib import Path

def Create(filename):
    root = ET.Element("students")
    tree = ET.ElementTree(root)
    with open(filename, "wb") as file:
        tree.write(file)

def Append(filename, name, email):
    student = ET.Element("student")
    ET.SubElement(student, "name").text = name
    ET.SubElement(student, "email").text = email

    root = ET.ElementTree(file=filename).getroot()
    root.append(student)
    with open(filename, "wb") as file:
        ET.ElementTree(root).write(file)

def List(filename):
    print()
    tree = ET.ElementTree(file=filename)
    for student in tree.findall('student'):
        for node in student.getiterator():
            if node.text:
                print("{}: {}".format(node.tag, node.text))
        updated = student.attrib.get('updated')
        if updated:
            print(">>> Last updated at", updated)

def Update(filename, name, newEmail):
    tree = ET.ElementTree(file=filename)
    root = tree.getroot()
    for elem in tree.iterfind('student[name="{}"]'.format(name)):
        elem.find("email").text = newEmail
        elem.set('updated',
datetime.datetime.now().strftime("%d/%m/%y %H:%M:%S"))
    with open(filename, "wb") as file:
        ET.ElementTree(root).write(file)

def Delete(filename, name, all = False):
    removed = 0
    tree = ET.ElementTree(file=filename)
    root = tree.getroot()
    if all:
        for e in root.findall('student[name="{}"]'.format(name)):
            root.remove(e)
            removed += 1
    else:
        e = root.find('student[name="{}"]'.format(name))
        if e:
            root.remove(e)
            removed = 1
    with open(filename, "wb") as file:
        ET.ElementTree(root).write(file)
    return removed

if __name__ == "__main__":
    filePath =
str(Path(os.path.dirname(__file__)).parent.parent.joinpath('Temp',
'students2.xml'))
    Create(filePath)
    Append(filePath, "student1", "student1@email.com")
    Append(filePath, "student2", "student2_1@email.com")
    Append(filePath, "student3", "student3@email.com")
    Append(filePath, "student4", "student4@email.com")
    Append(filePath, "student2", "student2_2@email.com")
    Append(filePath, "student2", "student2_3@email.com")
    Append(filePath, "student2", "student2_4@email.com")
    Append(filePath, "student2", "student2_5@email.com")
    List(filePath)

    Update(filePath, "student1", "XXX@email.com")
    List(filePath)

    print("Removed:", Delete(filePath, "student3"))
    List(filePath)

    print("Removed:", Delete(filePath, "student2"))
    List(filePath)

    print("Removed:", Delete(filePath, "student2", True))
    List(filePath)

    print("Removed:", Delete(filePath, "studentXXX", True))
    List(filePath)

    print("Done!")
```

Conclusion

Next?

- [A Byte of Python](https://python.swaroopch.com/)
- [Google's Python Class](https://developers.google.com/edu/python/)
- [Codecademy - Learn Python](https://www.codecademy.com/learn/learn-python)
- [How to Think Like a Computer Scientist: Learning with Python 3](http://openbookproject.net/thinkcs/python/english3e/)
- [Think Python](http://greenteapress.com/wp/think-python-2e/)

QA

Python Fundamental

Python