Python 3

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!
- On terminal: $ python FILENAME.py
"""

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

Shebang

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

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

Comments

```
"""
Comments
"""

# This is a single line comment
# This is another single line comment

'''
This
is
a
multiline
comment
'''

"""
This
is
another
multiline
comment
"""

print("Done with comments!!")
```

Interactive Shell (REPL)*

```
$ python

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

$ exit()
```

@[1] @[3-5] @[7]

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

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

Rules

```
"""
Rules
- See: module keyword for the list of python reserved words
"""

# ************************************************************
# 1. One statement per line
# ************************************************************

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

print("Student 1"); print("Student 2") # valid but bad practice

"""
print("Student 1") print("Student 2") # Error
"""

# ************************************************************
# 2. Leading whitespaces (spaces and tabs) are used to
#    define the level and the grouping of statements
# ************************************************************

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

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

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

# ************************************************************
# 3. Split code with \
# ************************************************************

print("Laborum reprehenderit voluptate dolore adipisicing \
dolore aliqua magna voluptate commodo reprehenderit sit.")

# ************************************************************
# 4. Naming: ^[a-zA-Z_$][a-zA-Z_$0-9]*$
# ************************************************************

student_1 = "" # valid
_student_1 = "" # valid, can start with underscore

"""
2Count = 0 # Error: cannot start with numbers
"""

# ************************************************************
# 5. Case Sensitive
# ************************************************************

print("I´m case sensitive!") # OK

"""
Print("I´m case sensitive!") # Error: Python is case sensitive!
a = 0
print(A) # 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)
```

@[1-3] @[5] @[7-8] @[10-11]

input

```
"""
input
"""

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

Data types

Scalars

Integer

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

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

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

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

# Using constructors with base
print()
value = int("10")
print(value)
value = int("10", 2) # binary
print(value)
value = int("10", 8) # octal
print(value)
value = int("10", 16) # hexadecimal
print(value)
```

Float

```
"""
Float
"""

import sys

value = 2.123
print(value)

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

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

# Max. Value
print()
print("{0}".format(sys.float_info.max)) # presents most readable
print("{0:f}".format(sys.float_info.max)) # as a long float value
```

```
# Promotion to float
print()
value = (2 + 1) * 3 + 1 # value will be integer
print(value)
value = (2 + 1) * 3 + 1.0 # value will be float
print(value)
```

None

```
"""
None
- Absence of a value
"""

value = None

print(value == None)
print(value 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)

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)
```

Special values

```
"""
Special values
"""

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

Strings

Strings

```
"""
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
"""

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

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

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

print()
str1 = "SELECT * "
str2 = "FROM students "
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()
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()
print("isnumeric:", message.isnumeric())
```

```
print("isnumeric:", "123".isnumeric())

print()
print("lower:", message.lower())
print("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()
print("in:", str('Z' in message))

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

print()
print("strip :" + " Python ".strip() + "|")
print("lstrip:" + " Python ".lstrip() + "|")
print("rstrip:" + " Python ".rstrip() + "|")

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 sep, sep and after sep
print("Team1 VS Team2".partition(" VS "))

print()
```

```
t1, t2, t3 = "Team1 VS Team2".partition(" VS ")
print(t1, t2, t3)
t1, _, t2 = "Team1 VS Team2".partition(" VS ") # throwaway var.
print(t1, t2)

print()
student = "Student1;10;11;12".partition(";")
print("name:{} grades:{}".format(student[0], student[2]))
```

Operators

Aritmetic

```
"""
Aritmetic
"""

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

```
"""
Bitwise
"""

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

```
"""
Logical
"""

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
value = int(input("input a value: "))

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

if else

```
"""
if else
"""

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

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

if elif else

```
"""
if elif else
"""

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

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

Logical operators and nested if

```
"""
Logical operators and nested if
"""

grade1 = 15
grade2 = 6
avg_grade = (grade1 + grade2) / 2.0

if avg_grade >= 9.5 and grade1 >= 9.5 and grade2 >= 9.5:
```

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

print()
student_name = "Student 1"
is_student = False
if student_name == "Student 1" \
    or student_name == "Student 2" \
    or student_name == "Student 3":
    is_student = True
print(is_student)

print()
is_student = student_name in ["Student 1", "Student 2", "Student 3"]
print(is_student)
```

Chained operations

```
"""
Chained operations
"""

# chained assigment
grade1 = grade2 = grade3 = 15
print(grade1, grade2, grade3)

print()
# chained condition
x = 0 <= grade1 <= 20 # same as 0 <= grade and 0 <= grade
print(x)
x = grade1 == grade2 == grade3 == 15
print(x)
```

Conditional expressions (ternary)

```
"""
Conditional expressions (ternary)
"""

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

is_valid = False
value = 1 if is_valid 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:
    value = int(input("input a value: "))
    if value == NUMBER:
        print("Winner!!!")
        hit = True
    elif value > NUMBER:
        print("your number is greater")
    else:
        print("your number is lower")
```

while else

```
"""
while else
"""

NUMBER = 17
hit = False

while not hit:
    value = int(input("input a value: "))
    if value == NUMBER:
        hit = True
    elif value > 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))
        value = int(input("input a value: "))
        if value == NUMBER:
            print("Winner!!!")
            hit = True
        elif value > 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))
    value = int(input("input a value: "))
    if value == NUMBER:
        print("WINNER!!!!")
        break
    elif value > 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)
```

_

```
"""

_
"""

for _ in range(0, 50):
    print("Hello")
```

Functions

Funtions and empty blocks (pass)

```
"""
Funtions and empty blocks (pass)
```

```
"""

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

say_hello()

def test():
    """I will implement this function later..."""
    pass

say_hello()
test()
say_hello()
```

Parameters

```
"""
Parameters
"""

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

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

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

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

print()
value1 = 10
value2 = 12
print("Max ({}, {}): ".format(value1, value2))
print_max(value1, value2)
```

```
print()
say("Python", value1)
```

Scope

```
"""
Scope
"""

value = "Global Scope"

def write_v1():
    value = "Inner Scope"
    print(value)

def write_v2(value): # 1) by Value, 2) Overloading is not allowed
    print("Param. Value: " + value)
    Value = "Inner Scope"
    print(Value)

def write_v3():
    global value
    print("Value: " + value)
    value = "Inner Scope"
    print(value)

def outer():

    def inner():
        print("2")

    print("1")
    inner()

print(value)
write_v1()
print(value)

print()
write_v2(value)
print(value)

print()
write_v3()
print(value)
```

```
print()
outer()
```

Default values and named parameters

```
"""
Default values and named parameters
"""

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

def sum_and_multiply(value1, value2=0, times=1):
    val = value1 + value2
    multiply(times=times, value=val)

multiply("Python")
multiply("Python", 3)
multiply(3)
multiply(3, 3)

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

Variable number of arguments

```
"""
Variable number of arguments
"""

def concat_spaced(*names):
    return_value = ""
    for n in names:
        return_value += n + " "
    print(return_value)

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

```
    print(start)

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

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

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

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

return

```
"""
return
"""

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

def is_greater_v1(value1, value2):
    if value1 > value2:
        return True
    else:
        return False

def is_greater_v2(value1, value2):
    if value1 > value2:
        return True
    return False
```

```
def my_sqrt(value):
    if value > 0:
        result = value ** 0.5
        print("Square root of {} is {}".format(value, result))
        return
    print("value must be > 0")

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

print()
print(is_greater_v1(10, 5))
print(is_greater_v1(10, 11))

print()
print(is_greater_v2(10, 5))
print(is_greater_v2(10, 11))

print()
my_sqrt(2)
my_sqrt(-2)
```

lambda expressions

```
"""
lambda expressions
"""

f = lambda x: x ** (2 if x > 5 else 3)
for i in range(10):
    print(f(i))

points = [{ 'x' : 2 , 'y' : 3 }, { 'x' : 4 , 'y' : 1 }]
points.sort(key= lambda x: x['y'])
print(points)

pairs = [(1, "one"), (2, "two"), (3, "three"), (4, "four")]
pairs.sort(key=lambda x: x[1])
print(pairs)
pairs.sort(key=lambda x: x[0])
print(pairs)
```

Builtin Functions

abs

```
"""
abs
"""

values = [10, 11, 15, 36]

value = min(values) - max(values)
print(value)
print(abs(value))

print()
value = max(values) - min(values)
print(value)
print(abs(value))
```

any(iterable) | all(iterable)

```
"""
any(iterable) | all(iterable)
- Iterables: str, list, dict, range, tuple, set, bytes
"""

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

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

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

def is_div_by_3(val):
    return val % 3 == 0
```

```
print(is_div_by_3(9))
print(is_div_by_3(19))

print()
value = any(is_div_by_3(x) for x in range(1000, 1051))
print(value)

value = all(name == name.title() for name \
                in ["Student1", "Student2", "student3"])
print(value)
```

chr

```
"""
chr
"""

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

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

del

```
"""
del
"""

name = "Student 1"
print(name)

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

dir

```
"""
dir
```

```
"""

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

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

enumerate

```
"""
enumerate
"""

name = "Student1"
print(enumerate(name))
print(list(enumerate(name)))

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

print()
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])
```

filter

```
"""
filter
"""

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

values = list(range(10))
print("".join(str(x) for x in values))

values = filter(is_even, values)
print("".join(str(x) for x in values))

values = filter(lambda x: x % 3 == 0, list(range(10)))
print("".join(str(x) for x in values))
```

globals

```
"""
globals
"""

import pprint

myvar = 101

pprint.pprint(globals())

print(globals()["myvar"])
```

len

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

```
item = "Student 1"
print(len(item))

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

max

```
"""
max
"""

values = [10, 11, 15, 36]
print(max(values))

values = ["a", "ab", "AB", "_1", "0", "-1", "-"]
print(max(values))

print()
for i in values:
    print(i, ord(i[0]))
```

min

```
"""
min
"""

values = [10, 11, 15, 36]
print(min(values))

values = ["a", "ab", "AB", "_1", "0", "-1", "-"]
print(min(values))

print()
for i in values:
    print(i, ord(i[0]))
```

ord

```
"""
ord
"""

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

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

message1 = "Python is Great!"
message2 = "Python is GREAT!"

print(hash_me(message1))
print(hash_me(message2))

print_my_chars(message1)
print()
print_my_chars(message2)
```

range

```
"""
range
"""

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

# range (stop)
print(list(range(4)))
print_range(range(4))
print_range(range(10))

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

```
print_range(range(0, -20, -2))
```

sum

```
"""
sum
"""

values = [10, 10, 5, 25, 25, 25.0]
print(values)
print(sum(values))

values = list(range(10))
print(values)
print(sum(values))

values = ["10.5", "11", 12.6]
print(values)
print(sum([float(x) for x in values]))
```

type

```
"""
type
"""

def test():
    pass

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

vars

```
"""
vars
```

```
"""

import math

__version__ = 1.0 # attribute

print()
print(dir(math)) # list names from math module

print()
print(vars(math)) # list names and values

print()
print(vars()) # current module
```

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))
```

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 example

```
"""
Module example
"""

__version__ = 1.0 # builin attribute
__sprint__ = 7 # user attributes
some_value = 10

def my_sum(value1, value2):
    return value1 + value2

def daddy():
    return "Running from: " + __name__

if __name__ == "__main__":
    print(daddy())
    print(my_sum(some_value, some_value))
```

Import module example

```
"""
```

```
Import module example
"""

import _03_module

print(_03_module.my_sum(1.25, 3.2))
print(_03_module.daddy())
print(_03_module.__version__)
print(_03_module.__sprint__)
print(_03_module.some_value)
```

Import module example

```
"""
Import module example
"""

from _03_module import my_sum, __version__, __sprint__, some_value

print(my_sum(1.25, 3.2))

print(__version__)
print(__sprint__)
print(some_value)
```

from ... import *

```
"""
from ... import *
"""

from _03_module import *

print(my_sum(1.25, 3.2))

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

Packages

```
"""
Packages
- Packages(folders) 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.my_multiplier(15, 10))
print(Second.my_multiplier(15, 10))
```

Packages

```
"""
Packages
"""

from Package1 import ModuleTest
import Package2.ModuleTest

print(ModuleTest.my_multiplier(15, 10))
print(Package2.ModuleTest.my_multiplier(15, 10)) # full Path
```

Nested Packages

```
"""
Nested Packages
"""

from Package1 import ModuleTest
from Package1.Package1_1 \
    import ModuleTest as third # nested packages

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

DocString

```
"""
DocString
This is a dummy DocString for this module
- Execute (on script folder):
    $ python
    $ import _10_docString
    $ help (_10_docString)
    $ _10_docString.my_function(1, 2)
"""

def my_function(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(my_function(1, 2))
    print(my_function.__doc__)
```

Collections

Sequences

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

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

my_sequence = [10, 11, 13]
print(my_sequence[1])
print(my_sequence.index(10))
```

```
print(my_sequence.count(11))
print(" ".join(str(i) for i in reversed(my_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()
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 memory_address(obj):
    return hex(id(obj))

def print_memory(obj):
    print(memory_address(obj), ":", obj)

print(id(memory_address))

# integers are immutable
print()
integer_ = 5
print_memory(integer_)
integer_ += 1
print_memory(integer_)

print()
integer2_ = integer_
print(integer_ is integer2_) # identity equality

# tuples are immutable
print()
tuple_ = (1, 2, 3)
print_memory(tuple_)
tuple_ += (4, 5, 6)
```

```
print_memory(tuple_)

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

print()
list1 = [1, 2, 3]
list2 = [1, 2, 3]
print(list1 == list2)
print(list1 is list2)

print()
print(type(list_[0]))
print_memory(list_[0])
print_memory(list_)
list_[0] = 123
print_memory(list_[0])
print_memory(list_)
```

List

```
"""
List
"""

import math

def my_avg(myList):
    sum = 0
    for grade in myList:
        sum += grade
    return sum / len(myList)

def list_one_by_one_v1(myList):
    for i in range(len(myList)):
        print(myList[i])

def list_one_by_one_v2(myList):
```

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

def append_value(myList, value):
    myList.append(value)

def change_my_list(myList):
    myList.append(10)
    myList = [1, 1, 1, 1]
    print(myList)

grades = [10, 8, 4, 17]

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

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

print(10 in grades)

print()
grades.append(15)
print("Updated grades:", grades)
print("Updated avg:", my_avg(grades))

print()
append_value(grades, 12)
print("Updated grades:", grades)
change_my_list(grades)
print("Updated grades:", grades)

print()
list_one_by_one_v1(grades)
list_one_by_one_v2(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)

grades2 = [18, 9, 7] * 3
print()
print(grades2)
del grades2[3:]
print(grades2)

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

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

print()
print("index of 17: ", pos_grades.index(17))

print()
pos_grades.append(17)
print(pos_grades)
pos_grades.remove(17) # remove first
print(pos_grades)

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

Tuple

```
"""
Tuple
"""

def my_sum(values):
    sum_ = 0
    for item in values:
        if isinstance(item, int) or isinstance(item, float):
            sum_ += item
```

```
        else:
            sum_ += my_sum(item)
    return sum_

def my_len(values):
    sum_ = 0
    for item in values:
        if isinstance(item, int) or isinstance(item, float):
            sum_ += 1
        else:
            sum_ += my_len(item)
    return sum_

def sum_and_len(values):
    sum_ = 0
    len_ = 0
    for item in values:
        if isinstance(item, int) or isinstance(item, float):
            len_ += 1
            sum_ += item
        else:
            sum_ += my_sum(item)
            len_ += my_len(item)
    return (sum_, len_)

if __name__ == "__main__":

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

    first_year_grades = (15, 18, 12, 10)
    second_year_grades = (13, 13, 12, 11)

    empty_tuple = ()
    singleton_tuple = (2, ) # (2) is integer 2

    grades = (15, 20, first_year_grades, second_year_grades)
    # could be done without parentheses, but bad practice
    # grades = 15, 20, firstYearGrades, secondYearGrades

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

    del first_year_grades
    del second_year_grades

    temp = (10, ) * 3 # (10, 10, 10)
```

```
    grades += temp # concatenation

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

    print()
    print(19 in grades)

    print()
    print("Sum:", my_sum(grades))
    print("Grades:", my_len(grades))
    print("AVG:", my_sum(grades)/ my_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])

    a, b = sum_and_len(grades)
    print(a, b)

    _, b = sum_and_len(grades)
    print(b)
```

Tuple

```
"""
Tuple
"""

import _05_tuple_01 as t

def get_report(values):
    report = [] # empty list
    report.append(("sum", t.my_sum(values)))
    report.append(("len", t.my_len(values)))
    report.append(("Avg", t.my_sum(values) / t.my_len(values)))
    return report

first_year_grades = (15, 18, 12, 10)
second_year_grades = (13, 13, 12, 11)
```

```
grades = first_year_grades, second_year_grades

del first_year_grades
del second_year_grades

studentReport = get_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 are uniques and must be immutables.
"""

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
backup_student = student
student["john"] = "john@python.org"
print(student)
print(backup_student)

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

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]
}

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

print()
# after copy() the LISTs (complex object) of grades
# will refer to the same objects, so remain equals in both
backup_grades2 = grades.copy()
backup_grades2["student5"][0] = 3
print(backup_grades2)
print(grades)

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

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

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

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, student_grades in positive.items():
    print("Name: {} | Grades: {}".format(name, student_grades))
```

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

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

Files

Files

```
"""
Files
- open(FILE, ACCESS_MODE)
- ACCESS_MODE:
    - 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

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

Files

```
"""
Files
"""

import os

file = open(os.path.realpath(__file__), "r")
lines = file.readlines()
for line in lines:
    print(line, end="")
file.close()
```

Files

```
"""
Files
"""

import os

file = open(os.path.realpath(__file__), "r")
done = False
while True:
    line = file.readline()
    if len(line) == 0:
        break
    print(line, end="")

    if not done:
        print("FIRST LINE AGAIN")
        file.seek(0)
        done = True

file.close()
```

Files

```
"""
Files
"""
```

```
import os
import time
from pathlib import Path

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

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))

fileIn = open(os.path.realpath(__file__), "r")
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 (TypeError, ValueError):
    print("Something went wrong with values")
except:
    print("Something went wrong")
else: print("Everithings Ok.")
```

as

```
"""
as
"""

try:
    a = 1 / 0
except ZeroDivisionError as err:
    print("Error:", err)
except TypeError as err:
    print("Error:", err, "\nCould 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.")
```

Handler

```
"""
Handler
"""
```

```
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.")
```

Flow

```
"""
Flow
"""

import sys

def first(option):
    second(option)

def second(option):
    third(option)
    if option == 3:
        raise ZeroDivisionError("UPS")

def third(option):
    fourth(option)
    if option == 2:
        raise TypeError("TypeError")

def fourth(option):
    try:
        fifth(option)
    except AttributeError:
```

```
        print("AttributeError Caught in fourth")

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

try:
    first(1) # try with 1, 2, 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()
```

try import

```
"""
try import
"""

try:
    import cProfile as profiler
except:
    import profile as profiler

print(profiler.__all__)
```

Iterators

Iterator

```
"""
Iterator
"""

def get_next_value(data):
    try:
        return next(data)
    except StopIteration:
        return None
```

```
students = ["student1", "student2", "student3", "student4"]

iterator = iter(students)

while True:
    a = get_next_value(iterator)
    if a:
        print(a)
        continue
    break
```

Iterator

```
"""
Iterator
"""

class RangeAZ:

    def __init__(self, max):
        if isinstance(max, int) and 0 < max <= 26:
            self.current = 65
            self.max = self.current + max
        else: raise ValueError("max parameter must be in [1..26].")

    def __iter__(self):
        return self

    def __next__(self):
        if self.current == self.max:
            raise StopIteration
        value = chr(self.current)
        self.current += 1
        return value

def get_next_value(data):
    try:
        return next(data)
    except StopIteration:
        return None

iterator = iter(RangeAZ(26))

while True:
    a = get_next_value(iterator)
    if a:
```

```
            print(a, end=" ")
            continue
        break
```

yield

```
"""
yield
"""

def generator1():
    yield 1
    yield 2
    yield 3
    yield 4
    yield 5

def generator2():
    for i in range(1, 6):
        yield i

g = generator1()
print(next(g), end=" ")
print(next(g), end=" ")
print(next(g), end=" ")

print()
g = generator1()
for _ in range(5):
    print(next(g), end=" ")

print()
for value in generator1():
    print(value, end=" ")

print()
for value in generator2():
    print(value, end=" ")
```

Generators

```
"""
```

```
Generators
- lazyness, does not create previously the series of data,
  it provides the elements when necessary
"""

import random

def pop(count, iterable):
    """pop n elements from iterable"""
    counter = 0
    for item in iterable:
        if counter == count:
            return
        counter += 1
        yield item

def distinct(iterable):
    yielded = set()
    for item in iterable:
        if item not in yielded:
            yielded.add(item)
            yield item

def incremenetal_random_number():
    yielded = 1
    while True: # due to lazyness can model infinite series of data
        yielded += random.randint(0, yielded)
        yield yielded

def main():

    items = list("AAAAABBBBCCCDDE" * 3)
    print(items)

    print()
    for item in pop(4, items):
        print(item, end=" ")

    print()
    for item in distinct(items):
        print(item, end=" ")

    print()
    for item in pop(4, distinct(items)):
        print(item, end=" ")

    print()
    a = incremenetal_random_number()
    for i in range(3):
```

```
        print(next(a), end=" ") # could do this infinitely

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

Generator expressions

```
"""
Generator expressions
"""

items = (x*3 for x in range(1, 100) if x % 2 == 0)
for i in items:
    print(i, end=" ")

# Needs to generato new one because previous already finished.
# Python's iterator protocol only provides the next() method.
# and no method to reset an iterator.

print("\n")
items = (x*3 for x in range(1, 100) if x % 2 == 0)
items = list(items) # force list generation consume large quantity of
memory
print(items)

print()
sumValues = sum(x*3 for x in range(1, 100) if x % 2 == 0)
print(sumValues)
```

itertools

```
"""
itertools
See: https://docs.python.org/3.6/library/itertools.html
"""

from itertools import islice, count, chain

v = islice((x for x in count() if x % 3 == 0), 100)
for i in v:
    print(i, end=" ")
```

```
print("\n")
# force list generation
x = list(islice((x for x in count() if x % 3 == 0), 100))
print(x)

print()
students = ["Student1", "Student2", "Student3", "Student4",
"Student5"]
grades1 = [12, 14, 15, 15]
grades2 = [13, 14, 14, 14, 10]
v = chain(students, grades1, grades2)
for item in v:
    print(item, end=" ")

print("\n")
print(all(x>=10 for x in chain(grades1, grades2)))
```

Decorators

Decorators

```
"""
Decorators
See: https://stackoverflow.com/questions/739654/how-to-make-a-chain-
of-function-decorators/1594484#1594484
See:
https://pythonconquerstheuniverse.wordpress.com/2012/04/29/python-
decorators/
"""

def make_me_pretty(func):

    def wrapper(*args):
        print('-' * 20)
        print("{:^20}".format(func(*args)))
        print('-' * 20)

    return wrapper

@make_me_pretty
def add(x, y):
    return x + y

def main():
    add(3, 2)
```

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

Decorators

```
"""
Decorators
"""

import time
import random

def execTime(func):

    def wrapper():
        t1 = time.time()
        func()
        print("ExecTime: {}".format(time.time() - t1))

    return wrapper

@execTime
def dummy_function1():
    sum_ = 0
    for value in range(500):
        sum_ += value
    print("\nSum: ", str(sum_))

@execTime
def dummy_function2():
    mult = 1
    for value in range(1, 500):
        mult *= value
    print("Mult: ", str(mult))

@execTime
def main():
    dummy_function1()
    dummy_function2()

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

Decorators

```
"""
Decorators
"""

import logging
import sys
import random

logging.basicConfig(stream=sys.stdout, \
    format="%(asctime)s : %(levelname)s : %(message)s", \
    level=logging.DEBUG)
log = logging.getLogger("retry")

def log_me(func):

    def wrapper(arg):
        log.info("{}: {}".format(func.__name__, arg))
        func(arg)

    return wrapper

@log_me
def dummy1(arg):
    print("+++++", arg)

@log_me
def dummy2(arg):
    print("-----", arg)

@log_me
def dummy3(arg):
    print("*****", arg)

def main():
    functions = [dummy1, dummy2, dummy3]
    args = ["Server 1", "Server 2", "Server 3"]

    for i in range(20):
        functions[random.randint(0, len(functions)-1)]\
            (random.randint(0, len(args)-1))

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

Object Oriented Programming

OOP Part 1

Abstract Class - GeometricForm

```
"""
Abstract Class - GeometricForm
"""

class GeometricForm():
    """Abstract class"""

    def get_corner(self):
        """Return left-bottom most point of geometric form"""
        raise NotImplementedError("Must implement this")

    def __del__(self):
        print(self.__class__.__name__, self, "destroyed")
```

Class - Point

```
"""
Class - Point
"""

from _01_GeometricForm import GeometricForm

class Point(GeometricForm):
    """Class to define a point in space"""

    Counter = 0

    def __init__(self, x, y, name=None):
        self.__x = x # private
        self.__y = y # private
        self.name = name
        Point.Counter += 1

    def distance_from_origin(self):
        """Calculate distance from origin"""
        return ((self.__x ** 2) + (self.__y ** 2)) ** 0.5

    def get_point(self):
```

```
        return (self.__x, self.__y)

    # overides method from abstract class
    def get_corner(self):
        return self.get_point()

    def __str__(self):
        return "{}: {}x{}".format(self.name if self.name \
                        else "Not named", self.__x, self.__y)

    def __repr__(self):
        return "<{}, {}>".format(self.__class__.__name__, \
            self.__class__.__doc__)

    # rich comparison operators
    def __eq__(self, other):
        return self.distance_from_origin() ==
other.distance_from_origin()

    def __ne__(self, other):
        return self.distance_from_origin() !=
other.distance_from_origin()

    def __lt__(self, other):
        return self.distance_from_origin() <
other.distance_from_origin()

    def __le__(self, other):
        return self.distance_from_origin() <=
other.distance_from_origin()

    def __gt__(self, other):
        return self.distance_from_origin() >
other.distance_from_origin()

    def __ge__(self, other):
        return self.distance_from_origin() >=
other.distance_from_origin()

    # Operator overload
    # See: https://docs.python.org/2/library/operator.html
    def __add__(self, other):
        p = other.get_point()
        return (self.__x + p[0], self.__y + p[1])

    def __mul__(self, other):
        p = other.get_point()
        return (self.__x * p[0], self.__y * p[1])
```

```
    @classmethod
    def From_Point(cls, point):
        return cls(point[0], point[1], "From class method")

def main():
    # Attributes
    print(Point.__doc__)
    print(Point.distance_from_origin.__doc__)
    print(Point.__name__)
    print(Point.__dict__) # Dictionary containing the class's
namespace
    print(Point.__module__) # module name in which the class is
defined

    point1 = Point(2, 2)
    point1.name = "My first point"
    point2 = Point(1, 2)

    # call method from abstract class
    print()
    print(point1.get_corner()) # if not implemented raise error

    print()
    print(point1) # call to __str__
    print(repr(point1)) # call to __repr__

    print()
    print(point1.distance_from_origin())
    print(point1.get_point())

    # call to rich comparison operators
    print()
    print(point1 == point2)
    print(point1 != point2)
    print(point1 < point2)
    print(point1 <= point2)
    print(point1 > point2)
    print(point1 >= point2)

    # call to rich comparison operators
    print()
    point3 = Point.From_Point(point1 + point2)
    print(point3)

    print()
    print(Point.Counter)

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

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

Class - Point3D

```
"""
Class - Point3D
"""

import math
from _02_Point import Point

class Point3D(Point):

    def __init__(self, x, y, z):
        super().__init__(x, y)
        self.__z = z

    def distance_from_origin(self):
        p = super().get_point()
        return Point3D.DistanceFrom(0, 0, 0, p[0], p[1], self.__z)

    @staticmethod
    def DistanceFrom(x, y, z, x1, y1, z1):
        return math.sqrt(math.pow(x - x1, 2) + \
                         math.pow(y - y1, 2) + \
                         math.pow(z - z1, 2))

    def GetPoint(self):
        p = super().get_point()
        return (p[0], p[1], self.__z)

def main():

    print(Point3D.__bases__) # attribute base classes

    point = Point3D(1, 2, 3)

    print(isinstance(point, Point))

    print(issubclass(Point3D, Point))
    print(issubclass(Point, Point3D))

    print(point.distance_from_origin())

    print(point.get_point())
```

```
    print(point.get_corner())

if __name__ == "__main__":
    main()
    print("Done!")
```

Class - Rectangle

```
"""
Class - Rectangle
"""

import copy
from _01_GeometricForm import GeometricForm
from _02_Point import Point

class Rectangle(GeometricForm):

    def __init__(self, c, w, h):
        self.__corner = c
        self.__width = w
        self.__height = h

    def get_width(self):
        return self.__width
    def set_width(self, value):
        self.__width = value

    def get_height(self):
        return self.__width
    def set_height(self, value):
        self.__width = value

    Width = property(get_width, set_width)
    Height = property(get_height, set_height)

    def get_corner(self):
        return self.__corner.get_point()

    def __str__(self):
        return "Corner:{} Width:{} Height:{}".\
            format(self.__corner.get_point(), self.__width,
self.__height)

    def my_ids(self):
```

```
        print("self:", id(self))
        print("Corner:", id(self.__corner))
        print("Width:", id(self.__width))
        print("Height:", id(self.__height))

def main():

    rectangle = Rectangle(Point(1, 1), 1000, 1500)
    print(rectangle)
    print(rectangle.get_corner())

    print()
    rectangle.my_ids()

    print()
    rectangle2 = rectangle
    rectangle2.my_ids()

    print()
    rectangle3 = copy.copy(rectangle)
    rectangle3.my_ids()
    del rectangle3

    print()
    rectangle4 = copy.deepcopy(rectangle)
    rectangle4.Width = 20000
    rectangle4.my_ids()
    print()
    rectangle.my_ids()

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

Attributes

```
"""
Attributes
"""

from _02_Point import Point

def main():

    point1 = Point(1, 2)
    point2 = Point(2, 2)
```

```
    print(hasattr(point1, "color"))
    setattr(point1, "color", "red")
    print(hasattr(point1, "color"))

    print()
    color = getattr(point1, "color", "blue")
    print(color)
    color = getattr(point2, "color", "blue")
    print(color)

    delattr(point1, "color")
    print(hasattr(point1, "color"))

if __name__ == "__main__":
    main()
    print("Done!")
```

Decorators

```
"""
Decorators
Class - PointExt1
See: http://pythoncentral.io/difference-between-staticmethod-and-
classmethod-in-python/
"""

from _02_Point import Point

class PointExt1(Point):

    def __init__(self):
        super().__init__(0, 0)
        self.__temp = 0

    # instance method
    def instance_method(self, x):
        self.__temp = x
        print("InstanceMethod: {}".format(self.__temp))

    @classmethod
    def class_method(cls, x):
        """
        class of the object instance is implicitly passed as
argument.
```

```
        """
        print("ClassMethod: {} {}".format(cls.__name__, x))

    @staticmethod
    def static_method(x):
        """
        self (instance) and cls (class) are not
        implicitly passed as arguments.
        """
        print("StaticMethod: {} {}".format(x, Point.Counter))

def main():

    tempInstance = PointExt1()
    tempInstance2 = PointExt1()
    tempInstance.instance_method(10)
    tempInstance2.instance_method(15)

    print()
    tempInstance.class_method(10)
    PointExt1.class_method(10)

    print()
    tempInstance.static_method(10)
    PointExt1.static_method(10)

if __name__ == "__main__":
    main()
    print("Done!")
```

Properties - Getter and Setter

```
"""
Properties - Getter and Setter
Class - PointExt2
"""

from _02_Point import Point

class PointExt2(Point):

    def __init__(self, x, y):
        super().__init__(x + 2, y + 2)
        self.__color = None
```

```
    def get_color(self):
        return self.__color

    def set_color(self, value):
        self.__color = value

    Color = property(get_color, set_color)

def main():

    point = PointExt2(2, 2)
    print(point.Color)
    point.Color = "red"
    print(point.Color)

if __name__ == "__main__":
    main()
    print("Done!")
```

Properties - Decorators

```
"""
Properties - Decorators
Class - PointExt3
"""

from _02_Point import Point

class PointExt3(Point):

    def __init__(self, x, y):
        super().__init__(x + 2, y + 2)
        self.__color = None

    @property
    def Color(self):
        return self.__color

    @Color.setter
    def Color(self, value):
        self.__color = value

def main():

    point = PointExt3(2, 2)
```

```
    print(point.Color)
    point.Color = "red"
    print(point.Color)

if __name__ == "__main__":
    main()
    print("Done!")
```

OOP Part 2

Owner

```
"""
Owner
"""

class Owner():

    def __init__(self, name=None):
        self._name = name

    def get_name(self):
        return self._name

    def __str__(self):
        return "My name is " + self._name

if __name__ == "__main__":
    person = Owner("Jack")
    print(person)
```

Animal Class

```
"""
Animal Class
"""

from _01_Owner import Owner

class Animal:
    """Animal Class"""
```

```
    Counter = 0

    # initializer
    def __init__(self, ownby, name="A. Doe", age = 0):
        self._name = name
        self._walked = 0
        self._owner = ownby
        self._age = age
        Animal.Counter += 1

    def _my_iteration(self, text):
        print(self._name, ":", text)

    # Instance Methods
    def speak(self):
        self._my_iteration("GRRRRR!!")

    def get_name(self):
        return self._name

    def get_age(self):
        return self._age

    def has_owner(self):
        return self._owner

    def get_owner_name(self):
        if self.has_owner():
            return self._owner.get_name()
        raise UnboundLocalError

    def _my_default_tag(self, name, owner, age):
        print("*" * 25)
        print("Name:", name)
        print("Owner:", owner)
        print("Age", age)
        print("*" * 25)

    def get_tag(self, maker=None):
        if not maker:
            maker = self._my_default_tag
        return maker(self._name, self.get_owner_name(), self._age)

    def walk(self, steps):
        self._walked += steps
        print(self._name, "walked", steps, "steps")

    def escaped(self):
        Animal.Counter -= 1
```

```
    @classmethod
    def Which(cls):
        return cls.__name__

    @staticmethod
    def Count():
        return Animal.Counter

    def __del__(self):
        print("I was destroyed")

    def __str__(self):
        return "My name is {} and i walked {}
steps".format(self._name, self._walked)

    def __eq__(self, other):
        if self._name != other.get_name() or \
            self._age != other.get_age() or \
            self.get_owner_name() != other.get_owner_name():
            return False
        return True
```

OOP - Examples

```
"""
OOP - Examples
"""

from _01_Owner import Owner
from _02_Animal import Animal

owner1 = Owner("Jack")
owner2 = Owner("John")

animal1 = Animal(None)
animal2 = Animal(owner2, "Rufus", 10)

print()
print(type(owner1))
print(type(animal1))

print()
print(isinstance(animal1, Owner))
print(isinstance(animal1, Animal))
```

```
animals = [animal1, animal2]

print()
for a in animals:
    a.speak()

print()
for a in animals:
    Animal.speak(a)

print()
for a in animals:
    a.get_name()

print()
# everything is "public"
for a in animals:
    print(a._name) # Bad pratice

print()
for a in animals:
    try:
        print("{:15}: {}".format(a.get_name(), a.get_owner_name()))
    except UnboundLocalError:
        print("{:15}: {}".format(a.get_name(), "No Owner"))

# Bad practice, call GetOwner() instaed
# Law of Demeter: Only talk to your immediate friends.
# See: https://en.wikipedia.org/wiki/Law_of_Demeter
print(animal2._owner.get_name())

print()
for a in animals:
    try:
        a.get_tag()
    except UnboundLocalError:
        print("{:15}: {}".format(a.get_name(), "No TAG"))

def new_tag_maker(name, owner, age):
    print("-" * 30)
    print("My name is {} and i am {} years old.".format(name, age))
    print("Please contact: {}".format(owner))
    print("-" * 30)

print()
for a in animals:
    try:
        a.get_tag(new_tag_maker)
```

```
    except UnboundLocalError:
        print("{:15}: {}".format(a.get_name(), "No TAG"))
```

OOP - Examples

```
"""
OOP - Examples
"""

import _02_Animal as A
import _01_Owner as O

dog1 = A.Animal(None)
dog2 = A.Animal(O.Owner("Jack"), "Rufus", 15)

print(dog1) # call  __str__ from animal
print("Pack: ", A.Animal.Count())

dog1.walk(10)
dog2.walk(25)

dog1.walk(25)
dog1.speak()
dog1.walk(125)

dog1.escaped()
del dog1  # call  __del__ from animal

print("Pack: ", A.Animal.Count())

print()
print(A.Animal.__doc__)
print(A.Animal.walk.__doc__)

print()
dog3 = A.Animal(O.Owner("Jack"), "Rufus", 15)
print(id(dog2))
print(id(dog3))
print(dog2 == dog3) # uses __eq__ from animal
print(dog2 is dog3)
```

Dogs and Cats

```
"""
Dogs and Cats
"""

from _02_Animal import Animal

class Dog(Animal):

    def __init__(self, ownby, name="A. Doe", age=0):
        super().__init__(ownby, name, age)

    def speak(self):
        print(self._name, ": AU!!! AU!! AU!!")

    def sniff(self):
        print(self._name, " is sniffing!")

class Cat(Animal):

    def speak(self):
        print(self._name, ": MIAU!!! MIAU!! MIAU!!")
```

OOP - Examples

```
"""
OOP - Examples
"""

from _02_Animal import Animal
from _05_Dogs_and_Cats import Dog, Cat

A1 = Dog("Rufus")
A2 = Dog("Bobby")
A3 = Cat("Kitty")
A4 = Cat("Diamond")
A5 = Animal("T-REX")

# call to class method
print("Class: ", A1.Which())
print("Class: ", A2.Which())
print("Class: ", A3.Which())
print("Class: ", A4.Which())
print("Class: ", A5.Which())

print()
```

```
print("Pets: ", Animal.Count())

print()
A1.walk(10)
A2.walk(25)
A3.walk(5)
A4.walk(15)

print()
pets = [A1, A2, A3, A4, A5]
for animal in pets:
    if isinstance(animal, Dog):
        animal.sniff()
    animal.speak()
```

Class - Chipped

```
"""
Class - Chipped
"""

class Chipped:

    def __init__(self, chipNumber):
        self.chip = chipNumber

    def chip(self):
        print(self.chip)

    def get_name(self):
        return "Chipped with " + str(self.chip)
```

Multiple inheritance

```
"""
Multiple inheritance
Class - NewDog
"""

from _07_Chipped import Chipped
from _05_Dogs_and_Cats import Dog
```

```
class NewDog(Dog, Chipped):

    def __init__(self, name, chipNumber):
        Dog.__init__(self, None, name)
        Chipped.__init__(self, chipNumber)

    def Print(self):
        print(self.get_name())
        print(self.chip)

        print(Dog.get_name(self))
        print(Chipped.get_name(self))

if __name__ == "__main__":
    A1 = NewDog("Rufus", "XPTO0001")
    A1.Print()
```

Data Structures

Class - Location

```
"""
Class - Location
"""

import math

class Location():

    def __init__(self, name, value=0):
        self.name = name
        self.value = value

    def __str__(self):
        return str(self.name).title()

    def __gt__(self, other):
        return self.name > other.name
```

Class - Node

```
"""
```

```
Class - Node
"""

class Node:

    def __init__(self, obj=None, next=None):
        self.__obj = obj
        self.next = next

    def __str__(self):
        return str(self.__obj)
```

Class - LinkedList

```
"""
Class - LinkedList
"""

from _01_Location import Location
from _02_Node import Node

class LinkedList():

    def __init__(self):
        self.head = None
        self.length = 0

    def add(self, obj):
        self.head = Node(obj, self.head)
        self.length += 1

    def print(self):
        node = self.head
        while node:
            print(node, end=" ")
            node = node.next
        print()

    def _print_reversed(self, node):
        if node:
            self._print_reversed(node.next)
            print(node, end=" ")

    def print_reversed(self):
        self._print_reversed(self.head)
```

```
def main():

    linked_list = LinkedList()
    linked_list.add(Location("porto"))
    linked_list.add(Location("Aveiro"))
    linked_list.add(Location("lisboa"))

    linked_list.print()
    print()
    linked_list.print_reversed()

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

Stack

```
"""
Stack
"""

from _01_Location import Location

class StackADT():

    def pop(self):
        raise NotImplementedError("Must implement this")

    def push(self):
        raise NotImplementedError("Must implement this")

    def is_empty(self):
        raise NotImplementedError("Must implement this")

class Stack(StackADT):

    def __init__(self):
        self.__items = []

    def pop(self):
        return self.__items.pop()

    def push(self, obj):
        self.__items.append(obj)

    def is_empty(self):
        return self.__items == []
```

```
def main():

    stack = Stack()
    stack.push(Location("porto"))
    stack.push(Location("Aveiro"))
    stack.push(Location("lisboa"))

    while not stack.is_empty():
        print(stack.pop(), end=" ")

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

Queue

```
"""
Queue
"""

from _01_Location import Location
from _02_Node import Node

class QueueADT():

    def remove(self):
        raise NotImplementedError("Must implement this")

    def insert(self):
        raise NotImplementedError("Must implement this")

    def is_empty(self):
        raise NotImplementedError("Must implement this")

class Queue1(QueueADT):

    def __init__(self):
        self.Items = []

    def insert(self, obj):
        self.Items.append(obj)

    def remove(self):
        return_value = self.Items[0]
        del self.Items[0]
        return return_value
```

```
    def is_empty(self):
        return self.Items == []

class Queue2(QueueADT):

    def __init__(self):
        self.length = 0
        self.head = None

    def insert(self, obj):
        node = Node(obj)
        if self.is_empty():
            self.head = node
        else:
            last = self.head
            while last.next:
                last = last.next
            last.next = node
        self.length += 1

    def remove(self):
        return_value = self.head
        self.head = self.head.next
        self.length -= 1
        return return_value

    def is_empty(self):
        return self.length == 0

class Queue3(QueueADT):

    def __init__(self):
        self.length = 0
        self.head = None
        self.tail = None

    def insert(self, T):
        node = Node(T)
        if self.is_empty():
            self.head = self.tail = node
        else:
            self.tail.next = node
            self.tail = node
        self.length += 1

    def remove(self):
        return_value = self.head
        self.head = self.head.next
```

```
        self.length -= 1
        if self.is_empty():
            self.tail = None
        return return_value

    def is_empty(self):
        return self.length == 0

def testQ(queue):
    queue.insert(Location("porto"))
    queue.insert(Location("Aveiro"))
    queue.insert(Location("lisboa"))
    while not queue.is_empty():
        print(queue.remove(), end=" ")

def main():

    testQ(Queue1())
    print()

    testQ(Queue2())
    print()

    testQ(Queue3())
    print()

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

Priority Queue

```
"""
Priority Queue
"""

from _01_Location import Location
from _05_Queue import Queue1

class PriorityQueue(Queue1):

    def remove(self):
        pos = 0
        for i in range(1, len(self.Items)):
            if self.Items[i].value > self.Items[pos].value:
                pos = i
        return_value = self.Items[pos]
```

```
        del self.Items[pos]
        return return_value

def testPQ(queue):
    queue.insert(Location("porto", 216405))
    queue.insert(Location("Aveiro", 78455))
    queue.insert(Location("lisboa", 506892))
    while not queue.is_empty():
        print(queue.remove(), end=" ")

def main():
    testPQ(PriorityQueue())
    print()

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

Tree

```
"""
Tree
"""

from _01_Location import Location

class TreeADT():

    def insert(self):
        raise NotImplementedError("Must implement this")

    def traverseInorder(self):
        raise NotImplementedError("Must implement this")

    def traversePreorder(self):
        raise NotImplementedError("Must implement this")

    def traversePostorder(self):
        raise NotImplementedError("Must implement this")

class Tree(TreeADT):

    class Node():

        def __init__(self, obj):
            self.obj = obj
            self.left = None
```

```
            self.right = None

        def __str__(self):
            return str(self.obj)

    def __init__(self):
        self.root = None

    def insert(self, obj, node=None):
        if not self.root:
            self.root = Tree.Node(obj)
        else:
            if node is None:
                node = self.root
            if node.obj > obj:
                if node.left is None:
                    node.left = Tree.Node(obj)
                else:
                    self.insert(obj, node.left)
            else:
                if node.right is None:
                    node.right = Tree.Node(obj)
                else:
                    self.insert(obj, node.right)

    def traverseInorder(self):
        self._traverseInorder(self.root)

    def traversePreorder(self):
        self._traversePreorder(self.root)

    def traversePostorder(self):
        self._traversePostorder(self.root)

    def _traverseInorder(self, node):
        if node is not None:
            self._traverseInorder(node.left)
            print(node.obj)
            self._traverseInorder(node.right)

    def _traversePreorder(self, node):
        if node is not None:
            print(node.obj)
            self._traversePreorder(node.left)
            self._traversePreorder(node.right)

    def _traversePostorder(self, node):
        if node is not None:
            self._traversePreorder(node.left)
```

```
            self._traversePreorder(node.right)
            print(node.obj)

def main():
    tree = Tree()
    tree.insert(Location("porto"))
    tree.insert(Location("Felgueiras"))
    tree.insert(Location("Aveiro"))
    tree.insert(Location("Alijo"))
    tree.insert(Location("lisboa"))
    tree.insert(Location("beja"))
    tree.insert(Location("evora"))
    tree.insert(Location("braga"))

    tree.traverseInorder()

    print()
    tree.traversePreorder()

    print()
    tree.traversePostorder()

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

Unit Test

unittest

```
"""
unittest
See: https://docs.python.org/3.6/library/unittest.html
See: https://jeffknupp.com/blog/2013/12/09/improve-your-python-
understanding-unit-testing/
"""

import unittest

class TextAnalysisTests(unittest.TestCase):

    # test cases begin with test_
    def test_function_runs(self):
        """The function run? aka 'Smoke test'"""
        multiply()
```

```
if __name__ == "__main__":
    unittest.main()
```

unittest

```
"""
unittest
"""

import unittest

class TextAnalysisTests(unittest.TestCase):

    def test_function_runs(self):
        multiply()

def multiply():
    pass

if __name__ == "__main__":
    unittest.main()
```

unittest

```
"""
unittest
"""

import unittest
from pathlib import Path
import os

class TextAnalysisTests(unittest.TestCase):

    # Fixture, called before each test
    def setUp(self):
        self.a = 10
        self.b = 11
        print(">", end="")

    # Fixture, called after each test
    def tearDown(self):
```

```
        del self.a, self.b
        print("<", end="")

    def test_function_runs(self):
        print("?", end="")
        multiply(self.a, self.b)

def multiply(value1, value2):
    print("!", end="")
    return value1 * value2

if __name__ == "__main__":
    unittest.main()
```

unittest

```
"""
unittest
"""

import unittest
from pathlib import Path
import os

class TextAnalysisTests(unittest.TestCase):

    def setUp(self):
        self.a = 100
        self.b = 10

    def tearDown(self):
        del self.a, self.b

    def test_function_runs(self):
        self.assertEqual(multiply(self.a, self.b), 1000)

    def test_Division_runs(self):
        self.assertEqual(divide(self.a, self.b), 10)

    def test_TenLenghtString_runs(self):
        assert len(create_lenght10_string()) == 10

    def test_Division_ZeroException(self):
        with self.assertRaises(ZeroDivisionError):
            divide(1, 0)
```

```
def multiply(value1, value2):
    return value1 * value2

def divide(value1, value2):
    return value1 / value2

def create_lenght10_string():
    return 'A' * 10

if __name__ == "__main__":
    unittest.main()
```

assert

```
'''
assert
raise an AssertionError if False
'''

MaxInField = 9
Team1 = set(range(1, 12))
Team2 = {1, 2, 4, 7, 5, 7, 11, 10, 23}
Team3 = {1, 2, 4, 7, 5, 8, 11, 10, 23, 13, 14, 15}

print(Team1, len(Team1))
print(Team2, len(Team2))
print(Team3, len(Team3))

try:
    assert len(Team1) <= MaxInField
    assert len(Team2) <= MaxInField
    print('Play the game!!')
except AssertionError:
    print('Teams are not well defined.')
```

Standard Library

The Python Standard Library

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

Modules

configparser

```
"""
configparser
"""

import configparser
import os
from pathlib import Path

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

def create_config_file(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")
    save_config_file(filepath, conf)

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

def get_setting(filepath, section, setting):
    return get_config_file(filepath).get(section, setting)

def set_setting(filepath, section, setting, value):
    conf = get_config_file(filePath)
    conf.set(section, setting, value)
    save_config_file(filepath, conf)

def remove_setting(filepath, section, setting):
    conf = get_config_file(filepath)
    conf.remove_option(section, setting)
    save_config_file(filepath, conf)

if __name__ == "__main__":

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

```
    create_config_file(filePath)

    set_setting(filePath, "Settings", "font", "Helvetica")
    print(get_setting(filePath, "Settings", "font_info"))

    set_setting(filePath, "Settings", "XXXXX", "XXXXX")
    print(get_setting(filePath, "Settings", "XXXXX"))

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

cProfile

```
"""
cProfile
"""

import time

def Rabbit():
    print("Rabbit...")

def Turtle():
    time.sleep(3)
    print("Turtle...")

def Dog():
    time.sleep(1)
    print("Dog...")

def main():
    Rabbit()
    Turtle()
    Dog()

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

cProfile

```
"""
cProfile
- Columns:
```

```
    - ncalls- number of calls made
    - tottime- total time spent in current function
    - percall- tottime divided by ncalls
    - cumtime- cumulative time spent in current function and
subfunctions
    - percall- cumtime divided by primitive calls
    - fileName:lineNumber(function)
"""

import cProfile
import os
from pathlib import Path
import _cProfile_01

cProfile.run("_cProfile_01.main()")
filePath =
str(Path(os.path.dirname(__file__)).parent.parent.joinpath("Temp",
"cProfile_output.txt"))

cProfile.run("_cProfile_01.main()", filePath)

# On Command Line:
# $ python -m cProfile -o ..\..\Temp\output.txt .\_cProfile_01.py
# output.txt will be binary, pstats needed

print("Done!")
```

pstats

```
"""
pstats
"""

import pstats
import os
from pathlib import Path

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

p = pstats.Stats(filePath)
p.strip_dirs().sort_stats(-1).print_stats()
```

csv

```
'''
csv
'''

import os
import csv
import random
from pathlib import Path

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

def write_CSV_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 read_CSV(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)]
    ]
```

```
    write_CSV(filePath, Students)
    read_CSV(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()
    write_CSV_DICT(filePath, newStudents, header)
    read_CSV(filePath)
```

datetime

```
"""
datetime
See: http://strftime.org/
"""

import datetime

data1 = datetime.date(2017, 6, 10)
data2 = datetime.datetime(2017, 6, 10, 11, 30, 0)

print(data1)

print(data2)
print(data2.year)
print(data2.month)
print(data2.day)
print(data2.hour)
print(data2.minute)
print(data2.second)

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

print(current_date.strftime("%d/%m/%y")) # str from date
print(current_date.strftime("%Y %b %d %a"))
print(current_date.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 = current_date - 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"))
```

dis

```
"""
dis
See: https://docs.python.org/3/library/dis.html
"""

import dis

def swap1():
    a = 5
    b = 4
    a, b = b, a

def swap2():
    a = 5
    b = 4
    c = a
    a = b
    b = c

print("swap1():")
dis.dis(swap1)

print("swap2():")
dis.dis(swap2)
```

keyword

```
"""
keyword
"""

import keyword

print(keyword.kwlist)
```

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.pow(2, 3))

print(math.factorial(5))

print(math.floor(3.5))
print(math.ceil(3.5))

print(math.pi)

pprint(vars(math))
```

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.")
```

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)
```

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 item in os.environ.items():
    print(item)

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)
```

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))
```

pprint (Pretty Print)

```
"""
pprint (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)
```

random

```
"""
random
"""

import random

start = 0
end = 100
seed = 123

rnd = random.Random(seed) # random object with seed
print(rnd.random())
print(random.random())

print(random.randint(start, end))
print(random.randrange(end))
```

```
print(random.randrange(start+2, end))
print(random.randrange(start+2, end, step=2))

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)]
]

print(students)
random.shuffle(students)
print(students)
```

smtplib

```
"""
smtplib
"""

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")
```

sys

```
"""
sys
"""

import sys
from pprint import pprint

def my_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: ", my_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
- On terminal: Use powershell command $LastExitCode to see error code
"""

import sys

def main():
    if len(sys.argv) < 2:
        sys.exit('Command line arguments are missing')
    try:
        result = int(sys.argv[1])
        print(str(result * 5))
        return 0
    except:
        return 1

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

this

```
"""
this
"""

import this
```

threading

```
'''
threading
- Execute on terminal
'''

import random
import time
from threading import Thread

class MyThreadClass(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 main():
    for i in range(1, 6):
        name = "Thread n:{}".format(i)
        my_thread = MyThreadClass(name)
        my_thread.start()

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

Time

```
'''
```

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

import time

def sleep_inc(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

time1 = time.time() #ticks
print(time1, "ticks")
time1 = time.localtime(time1) #struct
print(time1)
print(time1[0])
time1 = time.asctime(time1)
print(time1)

sleep_inc(5)
```

tkinter

```
"""
tkinter
"""

import tkinter
from tkinter import messagebox

class App:

    def __init__(self, parent):
        frame = tkinter.Frame(parent, width=125, height=75)
        frame.pack()
        frame.pack_propagate(0) # prevent frame skink to content

        tkinter.Button(frame, text="Bt1", \
```

```
            command=lambda: self._inform("first")).\
            pack(side=tkinter.LEFT)
        tkinter.Button(frame, text="Bt2", \
            command=lambda: self._inform("second")).\
            pack(side=tkinter.LEFT)
        tkinter.Button(frame, text="QUIT", \
            fg="red", command=frame.quit).\
            pack(side=tkinter.RIGHT)

    def _inform(self, num):
        messagebox.showinfo("info", "You pressed the {}
button".format(num))

if __name__ == "__main__":
    root = tkinter.Tk()
    app = App(root)
    root.mainloop()
```

turtle

```
"""
turtle
"""

import turtle
import random

window = turtle.Screen()
window.bgcolor("lightgreen")
window.title("My first Turtle")
my_turtle = turtle.Turtle()
my_turtle.shape("turtle") # blank, circle, classic, square, triangle,
turtle
my_turtle.pensize(2)
my_turtle.speed(5)

def circle():
    my_turtle.pendown()
    my_turtle.color("red")
    my_turtle.begin_fill()
    my_turtle.circle(random.randint(-30, 30))
    my_turtle.end_fill()

for c in range(4):
    circle()
```

```
    my_turtle.color("green")
    my_turtle.penup()
    for i in range(10):
        my_turtle.stamp()
        my_turtle.forward(20)
    my_turtle.right(90)

my_turtle.hideturtle()
window.mainloop() # Wait for user input to close window
```

turtle

```
"""
turtle
"""

import turtle
import random

def bar(t, h, size, factor, color):
    t.color("black", color)
    t.begin_fill()
    t.left(90)
    t.forward(h * factor)
    t.write("{:2}".format(h))
    t.right(90)
    t.forward(size)
    t.right(90)
    t.forward(h * factor)
    t.left(90)
    t.end_fill()
    t.color("black", "gray")

def separator(t, size):
    t.forward(size)

window = turtle.Screen()
window.title("Student grades")
window.bgcolor("white")

my_turtle = turtle.Turtle()
my_turtle.color("black", "gray")
my_turtle.pensize(1)
my_turtle.shape("blank")
```

```
my_turtle.penup()
my_turtle.setx(-125)
my_turtle.sety(-75)
my_turtle.pendown()

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)]
]

separator(my_turtle, 10)

for student in students:
    #myTurtle.write(student[0,])
    my_turtle.penup()
    my_turtle.sety(my_turtle.position()[1] - 20)
    my_turtle.write(student[0], move=False, align="left", font=
("Arial", 8, "normal"))
    my_turtle.sety(my_turtle.position()[1] + 20)
    my_turtle.pendown()
    for i in range(1, 4):
        bar(my_turtle, student[i], 10, 10, "green" if student[i] >=
9.5 else "red")
        separator(my_turtle, 5)
    separator(my_turtle, 10)
window.mainloop()
```

turtle

```
"""
turtle
"""

import turtle

wn = turtle.Screen()
wn.title("Turtle 3")
wn.bgcolor("aqua")
```

```
my_turtle = turtle.Turtle()
my_turtle.shape("turtle")
my_turtle.color("green", "lightgreen")

myTurtle2 = turtle.Turtle()
myTurtle2.shape("turtle")
myTurtle2.color("green", "green")

current = my_turtle

def swap():
    global current
    current = myTurtle2 if current == my_turtle else my_turtle

def up():
    current.forward(30)
def left():
    current.left(45)
def right():
    current.right(45)
def down():
    current.backward(30)

def quit():
    wn.bye()

def goto(x, y):
    my_turtle.goto(x, y)
    #wn.title("Turtle at {0}, {1}".format(x, y))

wn.onkey(swap, "s")
wn.onkey(up, "Up")
wn.onkey(left, "Left")
wn.onkey(right, "Right")
wn.onkey(down, "Down")
wn.onkey(quit, "q")
wn.onclick(goto)

wn.listen()
wn.mainloop()
```

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)
```

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.get_XML(url)
        self.get_students(students)

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

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

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

    def get_students(self, xml):
        self._enrolled = xml.getAttribute("enrolled")
        for node in xml.getElementsByTagName("student"):
            number = node.getAttribute("number")
            name = self.get_text(node.getElementsByTagName("name"))
            grades =
self.get_grades(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 display(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")
    display(filePath)

    update(filePath, "student1", "XXX@email.com")
    display(filePath)

    print("Removed:", delete(filePath, "student3"))
    display(filePath)

    print("Removed:", delete(filePath, "student2"))
    display(filePath)

    print("Removed:", delete(filePath, "student2", True))
    display(filePath)

    print("Removed:", delete(filePath, "studentXXX", True))
    display(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/)
- [Google Python Style Guide](https://google.github.io/styleguide/pyguide.html)

Python 3