Python

Basics

Variables are declared in snake_case

```
my_var = 5
```

There are no constants in python, we can write a variable in UPPERCASE_SNAKE_CASE to indicate that that variable shouldn't never have it's value changed.

```
MY_CONST = 4
```

Variable types

```
text = 'hola' # string
number = 100
decimal = 0.5
complx = 8j

people = ['Mario', 'Luigi'] # list (mutable)
numbers = (1, 2, 3) # tuple (immutable)
range_numbers = range(1, 1000) # 1 to 999

workers = {'Fernando':42, 'Timmy':32} # dictionary

unique_numbers = {1, 2, 2, 3, 3, 4} # set, removes repeated items and do not maintain the order of the items.
Mutable
unique_numbers2 = frozenset({1, 2, 2, 3, 3, 4}) # frozenset makes the set immutable

is_connected = True # boolean

empty = None
```

Type hinting

```
name: str = 'Mario'
number: int = 12
decimal: float = .2
```

Type conversion

```
my_str = '10'
number = int(my_str) # 10
type(number) # int

my_str = str(number) # '10'

bool(1) # True
float(number) # 10.0
```

Integers

```
a = 10
big_number = 100_000_000 # the underscores are only for clearer reading, they don't do anything
print(big_number) # 100000000
```

Operators

```
print(1 + 2)
print(1 - 2)
print(1 * 2)
print(1 / 2)
print(10 % 3) # modulus, the remainder of dividing one number by another, in this case 1
print(3 ** 2) # power
print(5 // 2) # floor division, returns abs value of division, in this case 2

a += 4 # a = a + 4
print(a == b) # equals
print(a != b) # not equals

print(2 > 3)
print(2 < 3)
print(2 >= 3)
print(2 <= 2)

print(3 > 4 or 4 > 3)
print(3 > 4 and 4 > 3)

print(not(5 < 2)) # true

a = 100.0
b = 1.0 * a

print(a is b)  # is => True if the compared items are in the same mem location, so False
print(a == b)  # they have same value so True
print(id(a))
print(id(b))


numbers = [1, 2, 3, 4, 5]
print(1 in numbers) # True
print(6 in numbers) # False
```

Strings

```
simple_quote = 'hello'
double_quote = "world"
mixed = 'hello "world"'
escape_quotes = 'hello \'lalala\'' # hello 'lalala'
multiline =
"""ey
ey
ey
"""

number = 9000

formatted_str = f'is over {number}!!!!' # is over 9000!!!!
```

Booleans

```
print(False == 0) # True
print(True == 1) # True
print(True == 2) # False
print(True + True) # 2
```

Lists

```
# list are a mutable collection of values
people: list[str] = ['Mario', 'Luigi', 'Peach', 'Toad']
print(len(people)) # length -> 4
print(people[1]) # Luigi
print(people[-1]) # Toad
print(people[0:4]) # 0 inclusive, 4 exclusive, so prints from index 0 to 3, all people
print(people[:4]) # same as 0:4
print(people[2:]) # from index 2 to 3 (the last one)
print('Luigi' in people) # True, 'Luigi' is a str present in the list
people[0] = 'Wario' # replaces Mario with Wario
print(people[0]) # Wario
people.insert(2, 'Tuigi') # inserts Tuigi in index 2, between Luigi and Peach
people.append('Waluigi') # inserts Waluigi as last index of the list

people2: list[str] = ['Sonic', 'Tails']
```

```python
people += people2 # appends the items of people2 to people
people.extend(people2) # same

people.remove('Mario') # removes Mario from list, if it not exists returns error
people.pop(2) # removes the element in index 2
people.pop() # removes last element
people.clear() # empties the list from elements
people.reverse() # reverse the order of the element in the list
people.sort() # sorts alphabetically
```

Tuples

```python
# tuples are like lists but immutable
people_tuple : tuple = ('Mario', 'Mario', 'Luigi','Sonic')
# casting to list
people_list : list[str] = list(people_tuple)
# casting to tuple
people_tuple : tuple = tuple(people_list)
people_tuple.index('Mario') # returns the first occurence, so 0
people_tuple.count('Mario') # 2

# unpacking to variables
a, b, c, d = people_tuple # so a is Mario, b is Mario, c is Luigi and d is Sonic
a, *b = people_tuple # a is Mario, and b will take the rest of the elements
```

Sets

```python
# sets are mutable, unordered collections that don't allow repeated items
items: set = {'apple', 'banana', 10, True, 'banana'} # second banana is ignored
print(items) # elements are printed in random order
items[0] = 10 # error, we cannot set the value of a given index, because sets are unordered
# empty set
people = set() # we cannot use people = {} because that would create a dictionary

items.add('orange') # adds one item to the set
items.update(['carrot', 15]) # adds multiple items to the set
items.remove('banana') # if not exists returns error
items.discard('banana') # if banana not exists there is no problem
items.clear()

items2: set = {'carrot', 15}

full_set = items.union(items2) # joins the two sets

full_set = items | items2 # same as union

items |= items2 # makes items the union of items and items 2

items3: set = {'apple', 'banana', 10, True}
items4: set = {'carrot', 15, 'apple', 'banana'}

items3.intersection_update(items4) # returns elements present in both sets {'apple', 'banana'}
items3.symmetric_difference_update(items4) # returns elements that are present in one of the sets, but not in
both {True, 10, 15, 'carrot'}

new_items = items3.symmetric_difference(items4) # if it isn't the update, it must be assigned to a new set
variable
```

Dictionaries

```python
users = {'user1':'Mario123', 'user2':'Luigi456'}
user1 = users['user1'] # return error if key doesn't exists
user1 = users.get('user1') # return None if key doesnt' exists
print(list(users.keys())) # returns only the keys, without values
print(list(users.values())) # returns only values, without keys
users['user1'] = 'Toad789' # changes the value for that key
x= list(users.items()) # return a list of tuples with each key-value pair as a tuple
users.update({'hello':123}) # appends kv pair to the dictionary
users.pop('user1') # removes user1 from dictionary
users.popitem() # removes last item from dictionary
users.clear() # empties the dict

nested_dict = {
    'user1':'Mario123',
    'user2':'Luigi456',
    'items': {'apple':10,
              'banana':20}
} # items is a nested dictionary inside the outer dictionary
print(users['items']['apple']) #returns 10

print(users.setdefault('user1', 'There is no key')) # if we provide a key that doesn't exist it will return
the value 'There is no key'
```

Control Flow

if else

```python
text = 'hello'

if text == 'hello':
    print('Bot: Hello!')
elif text == 'bye':
    print('Bot: Goodbye!')
else:
    print('I did not understand...')
```

for loop

```python
people = ['Mario','Luigi', 'Peach', 'Toad']

for person in people:
    print(f'Hello, {person}')

people_map = {'plumber':'Mario', 'cooler_brother':'Luigi'}

for k,v in people_map.items():
    print(f'Hello, i am {value}, and i am the {key}')

# for range if we need an index
for i in range(10): # from 0 to 9
    print(i)
```

while loop

```python
a = 0

while a < 10:
    print(a)
    a += 1
```

break and continue

```python
# print odd numbers from 1 to 50
i = 0
while True:
    if i % 2 == 0:
        i += 1
        continue # goes next loop

    print(i)
    i += 1

    if i >= 50:
        break # stops loop
```

pass

```
# print even numbers from 1 to 50
for i in range(100):
    if i > 0 and i % 2:
        pass # does nothing
    else:
        print(i)
```

loop else

```
for i in range(50):
    print(i)
else: # this else is going to execute only if nothing breaks the loop
    print('finished')
```

```
while i < 3:
    print(i)
    i += 1
else: # only executes when the while becomes false, not if a break stops the loop
    print('finished')
```

Inline Control Flow

shorthand if else

```
a = 4
b = 2
result = 'even' if a % b == 0 else 'odd'

print(True if a < b else False)
```

list comprehension

```
my_list = [i**2 for i in range(5)]
my_set = {i**2 for i in range(5)}
dict1 = {'a':2, 'b':4}
my_dict = {k:v**2 for k,v in dict1}
# adding conditional
my_list = [i**2 for i in range(5) if i%2==0]
```

map

```
# we pass a function and an iterable, and the function is executed for all the elements of the iterable
def my_function(element):
    return element * 2

my_iterable = [1, 2, 3]

my_mapped_list = map(my_function, my_iterable) # [2, 4, 6]
```

filter

```
# we pass a function which returns True or False, and an iterable, the elements of the iterable are processed
by the function and those that returns False are removed from the resulting iterable

def my_filtering_function(element):
    return element % 2 == 0

my_iterable = [1, 2, 3, 4]

my_filtered_list = filter(my_filtering_function, my_iterable) # [2, 4]
```

reduce (functools package)

```
from functools import reduce
# we pass a function and an iterable, through the execution of the function in all elements of the iterable,
the iterable is converted into an unique value

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

my_sum = reduce(add, [1, 2, 3, 4]) # 10 , this is similar to my_sum = add(add(add(1, 2), 3), 4)
```

Functions

```
def greet(name: str):
    print(f'Hey, {name}')

greet('Mario') # Hey, Mario
greet('Luigi') # Hey, Luigi

# with return
def double_the_number(number : int) -> int:
    return number * 2

# return collection
def return_collection(number : int) -> list[int]:
    my_list: List[int] = []
    my_list.extend([number, number+1, number+2])
    return my_list

my_collection = return_collection(2)
print(my_collection) # [2, 3, 4]

 # there cannot be parameters without default value after one parameter with default value has been declared
def default_value_parameter(number: int = 10):
    print(number)

default_value_parameter()   # 10
default_value_parameter(12) # 12

# multiple return
def double_and_triple(number: int) -> (int, int): # you could specify {int, int} or [int,int] or
{'double':int,'triple':int}
    return number * 2, number * 3 # returns a tuple, parenthesis are optional
    # return [number *2, number *3] # returns a list
    # return {number *2, number *3} # returns a set
    # return {'double':number * 2, 'triple':number * 3} returns a dictionary
```

lambda functions (anonymous functions)

```
# lambda arguments : expression
doubler = lambda x : x * 2
print(doubler(6)) # 12

adder = lambda x, y : x + y
print(adder(1,2)) # 3
```

Recursion

```
# a function calling himself in it's own code
def countdown(number: int):
    print(f'{number}!')
    if(number>0):
        countdown(number-1) # recursion
    else:
        print('launch!')

countdown(10)
```

*args y **kwargs

```
# args
def greet_people(lives:int, *people: str): # note the asterisk, *args must always be the last parameter,
except for **kwargs
    for name in people:
        print(f'Hello {name}!, you have {lives} lives left')

greet_people('Mario', 'Luigi', 'Toad')
```

```
# kwargs must always be the last parameter
def do_something(**kwargs):
    print(kwargs['name'])
    print(kwargs['age'])

    for k, v in kwargs.items():
        print(k)
        print(v)

do_something(name='Mario', age=10)
```

* and /

```
def standard_arg(arg): # arg can be positional or keyword (like kwargs)
    print(arg)

def pos_only_arg(arg, /):  # accept only positional arguments
    print(arg)

def kwd_only_arg(*, arg): # accept only keyword arguments.
    print(arg)
```

Classes

```
class my_class:
    def __init__(self, name, age): # constructor
        self.name = name
        self.age = age

    def __str__(): # toString
        return f'{self.name}, {self.age}'

    # you can add more method you define

my_instance_of_class = my_class('Monchito', 12) # instantiation of new object

# set property
my_instance_of_class.age = 20
# delete property
del my_instance_of_class.age
# delete instance
del my_instance_of_class
```

Error Handling

try except else finally

```
user_input = input('Enter a number:')

try:
    number = float(user_input)
    print(number)
except Exception as e:
    print(e)    # could not convert string to float: 'a'
```

```
user_input = input('Enter a number:')

try:
    number = float(user_input)
    result = number / 0
except ValueError:
    print('Please enter a valid number!')
except ZeroDivisionError:
    print('Please do not divide by 0!')
except Exception as e:
    print('Something went wrong!', e)
else
    print('all went well!') # only runs if no exception occurs
finally:
    print('program ended') # this runs no matter what happens, if there is an error or not
```

raise

```
is_connected: boolean = False

def connect_to_internet():
    if not is_connected:
        raise ConnectionError('No internet!')
    else:
        print('Connected to the internet!')

try:
    connect_to_internet()
except ConnectionError:
    print('There is no connection!')
except Exception as e:
    print(e)
```

Command Line Arguments

```
py app.py -e txt -n mydoc
py app.py -h # shows help describing all command line arguments

def main():
    parser: ArgumentParser = ArgumentParser()
    parser.add_argument('-e', '--extension', help='extension type', type=str)
    parser.add_argument('-n', '--name', help='file name',           type=str)
    args = vars(parser.parse_args())
```