Easy Python

Farhan Tanvir Utshaw · Follow
5 min read · Feb 9

Python is **dynamically typed** (type depends on assignment & given during runtime)↓

```
greeting = 'hello' # greeting is of type string
greeting = 123     # greeting is of type int
```

Python is a **strongly typed** language (checks data types when an operation is done on the variable) ↓

```
greeting = 123
print(greeting + "there") # doesn't convert int greeting 123 to str '123' au
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Name your variables

1. use snake_case for variable, function, package, module names: `sum_of_marks` , `forward_position` ,
2. Use PascalCase and noun for the class name: `Student` , `Person`
3. Use UPPER_CASE for constants: MAX_QUANTITY = 200

4. When the purpose of the function is to return something use noun: `area()` usage: `shape.area() # 12` not get_area()
5. Use verb when something is getting done inside the function: `empty_recyclebin()`

Variable, Identifier, literals

Variable: a location in the memory to hold data.

Identifier: the name associated with a variable or function to use those

Literal: A constant value typically stored in a variable.

salary = 12000 here *salary* is the variable name and *12000* is literal

Data types

Most commonly used data types: int, float, str, dictionary, set, list

int: is an integer (-100, -3, -2, 0, 1, 100, etc)

float: floating point numbers (100.12, 2.3, etc)

str: string data type i.e a sequence of characters (“John”, etc)

Input

Taking input (input function always assigns a string to the variable):

```
name = input("What's your name") # waits for user input
print(name) # name is of type string
number_of_courses = input("How many courses: ")
print(number_of_courses) # number_of_courses is string
number_of_courses = int(number_of_courses) # casting to int
volume_of_bottle = float(input('Enter the volume')) # casting immediately
```

Function

use `def` keyword then write the function name. The function body will be indented by four spaces or a tab equivalent.

```
varsity_name = 'UIU'

def say_hi(name, greeting):
   # global varsity_name
   varsity_name = 'BUET'
   print(f"{greeting} {name} from {varsity_name}")

say_hi(greeting='Hola', name='John')
print(varsity_name) # global
```

The output of the preceding program:

> *Hola John from BUET*
> *UIU*

When we are assigning a value to `varsity_name` inside `say_hi` function, the function will treat the `varsity_name` as local to the function. In order to utilize the global `varsity_name` variable, you must declare `global varsity_name` inside the function (uncommenting that statement).

Data types again

dictionary

insertion ordered (As of Python 3.7), **mutable**, doesn't support duplicate key, supports duplicate values

You want to get detailed information about something based on key information (preferably short). Then use a dictionary. The detail or intended information is value, and whatever info you got is the key.

For example, you want to know the details of a person with ID: 007. Then 007 is the key, the detail is the value. You put the detail against 007 key. Then whenever you want to know information you just give 007 to the storage, it will give you the detail. ↓

```
dict = {
    "007": "John Doe, a programmer"
```

```
}
```

```
print(dict["007"]) # John Doe, a programmer
print(dict["008"]) # KeyError
print(dict.get("009")) # None
print(dict.get("009", "Samantha")) # Samantha
del dict["007"] # deletes "007" entry
print("008" in dict) # True
for key in dict: # dictionary iterator gives the keys
    print(key, dict[key])
```

```
print(dict.keys()) # all the keys
print(dict.values()) # all the values
print(dict.items()) # key,val pairs
```

If the key doesn't exist, then we get KeyError when accessing the dictionary using a bracket. Using the get method gives None for the nonexistent key, the second argument can be used to get around None, this argument will be returned by get when the supplied key doesn't exist in the dictionary.

set

unordered, **mutable**, doesn't support duplicate content

The contents of the set must be immutable.

```
animal_types = {"human"}
animal_types.add("bird")
animal_types.update(["cat", "dog"])
animal_types.add("human") # doesn't add human again as it's there
animal_types.discard("dog") # discard dog from the set
print(animal_types)
plant_types = {"Herbs", "Shrubs"}
print(plant_types.union(animal_types)) # returns a new set
print(plant_types.intersection(animal_types))# returns a new set
print(animal_types.difference(plant_types))# returns a new set
```

list

insertion ordered, **mutable,** supports duplicate entry

```
pencil_box = [
    "Rubber", "Pencil", "Pen", "Scale", ("Sharpner", "Dirt"), 10000
]

print(pencil_box[2])
pencil_box.append("Leaflet") #  insertion at the end
pencil_box.insert(1, "Paper") # insertion at index 1
pencil_box.pop(1)  # remove element from index 1
for item in pencil_box: # for each loop
    print(item)
print(pencil_box.index('Scale')) # 3
```

tuple

ordered, immutable

```
courses = ("CSE", "EEE", "BBA")
print(courses[0]) # CSE
for course in courses:
    print(course)
```

Mutable vs Immutable

Mutable -> The original content can be changed

Immutable -> The original content can’t be changed

```
vowels = ["A", "E", "I", "O"]

letters = vowels
letters.append("U")
print(vowels).  # ['A', 'E', 'I', 'O', 'U']
print(id(vowels), id(letters)) # 4330599680 4330599680

person = "John"
student = "John"
print(id(person), id(student)) # 4331537008 4331537008
student = student + "son"
print(id(person), id(student)) # 4331537008 4331199280
```

Lists are mutable. Both vowels and letters are pointing to the same object. So, when we change through any of the variable, the original content is getting modified. [id function returns the address of the object]

On the other hand, the person and student are initially pointing to the same string literal “John”. But as strings are immutable, as soon as we start modifying a variable a new literal for that variable is getting created in the heap.

lambda expression

nameless aka anonymous functions

Can’t use lambda expression for multiline functions

> *lambda x1, x2, .. xn: x1 + x2 + … xn*

this expression sums all the values supplied to it

```
add_all = lambda x1, x2, x3 : x1 + x2 + x3
print(add_all(1, 2, 3)) # 6
```

lambda may return lambda too

```
straight_line_equation = lambda a, b, c: lambda x, y: a*x + b*y + c
print(straight_line_equation(1, 1, 1)(-2, 1))
```

Useful functions

sorting using sort/sorted

```
import functools

car_companies = [
    # company_name, revenue
    ("Volkswagen", "284"),
    ("Toyota", "270"),
```

```
    ("Stellantis", "182"),
    ("Mercedes", "156"),
    ("Ford", "151"),
    ("Honda", "147"),
    ("Tesla", "75")
]

car_companies.sort(key=lambda item: item[0])
print(car_companies)
car_companies.sort(key=functools.cmp_to_key(lambda item1, item2: float(item2
print(car_companies)
```

comparator function should return

1. a negative value for the item1 to be sorted before the item2
2. a positive value for the item1 to be sorted after the item2
3. 0 when both items are given equal priorities

map

```
companies = [
    # name, rev in B
    ("Amazon", 503),
    ("Amazon", 395),
    ("Alphabet", 283),
]

print(list(map(lambda x: x[1] * (10**9) , companies))) # [503000000000, 3950
```

zip

```
brands = [
    "Gucci",
    "Dior",
    "Chanel",
    "Louis Vuitton",
    "Hermes"
]

brand_net_worths = [
    18,
```

```
    154,
    30,
    440,
    4
]

print(list(zip(brands, brand_net_worths))) # [('Gucci', 18), ('Dior', 154),
```

pip, virtual environment

pip

the package manager for python

installs, updates packages not available in the standard python library

virtual environment

```
pip3 install virtualenv
virtualenv venv # create the environment named venv
source venv/bin/activate # activate the venv
pip list # all packages installed in the environment
pip freeze --local > requirements.txt # only the local dependencies
deactivate # goes out of the environment
# use specific python version using -p option
virtualenv -p /Library/Frameworks/Python.framework/Versions/3.10/bin/python3
source py310_env/bin/activate
pip install -r requirements.txt # install all dependencies specified in the
```

Class

```
class Car:
    def __init__(self, transmission, engine, tire):
        self.transmission = transmission
        self.engine = engine
        self.tire = tire

    def __eq__(self, car):
        return car.engine == self.engine and car.tire == self.tire and car.tr


    def check_if_equals(self, car):
        return car.engine == self.engine and car.tire == self.tire and car.tr
```

```
car = Car("CVT", "VVTI", "Pireli")
car2 = Car("CVT", "VVTI", "Pireli")

print()
print(car == car2)
```

Follow

Written by Farhan Tanvir Utshaw

6 Followers

CSE student at BUET

More from Farhan Tanvir Utshaw