OOP Using Python Day 2

February 1, 2025

1 OOP Using Python Day 2

1.1 Python Tips

1.1.1 One Line `if else`

[1]:
```
num = 5
print('greater') if num > 10 else print('smaller')
```

```
smaller
```

1.1.2 `split` & `join`

[2]:
```
names = ['Alice', 'Bob', 'Charlie']
'@'.join(names)
```

[2]:
```
'Alice@Bob@Charlie'
```

[3]:
```
myStr = 'this is a string'
myStr.split(' ')
```

[3]:
```
['this', 'is', 'a', 'string']
```

1.1.3 Swapping

[4]:
```
x = 1
y = 2

x, y = y, x
print(x, y)
```

```
2 1
```

1.1.4 `enumerate`

[5]:
```
langs = ['Python', 'Ruby', 'Perl', 'PHP']
print(list(enumerate(langs)))

for i, v in enumerate(langs):
    print(f"{i+1}: {v}")
```

```
[(0, 'Python'), (1, 'Ruby'), (2, 'Perl'), (3, 'PHP')]
1: Python
2: Ruby
3: Perl
4: PHP
```

1.1.5 is vs ==

The difference between `is` and `==` in Python is that `is` compares only the values of the variables, while `==` can convert the variables to the same type and then compare them.

Also when comparing iterable objects, `is` compares the memory address of the objects, while `==` compares the values of the objects.

[6]:
```
import warnings

warnings.filterwarnings("ignore", category=SyntaxWarning)
```

[7]:
```
print(True == 1)
print(True is 1)
l1 = [1, 2, 3]
l2 = [1, 2, 3]
print(l1 == l2)
print(l1 is l2)
```

```
True
False
True
False
```

1.1.6 any & all

[8]:
```
nums = [0, 1, 2, 3]

print(all(nums))
print(any(nums))
```

```
False
True
```

1.2 List Methods

1.2.1 pop, append, extend, and insert

[9]:
```
nums = [0, 1, 2, 3]

nums.pop()
print(nums)

nums.append(4)
```

```
print(nums)

nums.insert(2, 99)
print(nums)

nums.remove(99)
print(nums)

nums2 = [5, 6, 7]
nums.extend(nums2)
print(nums)
```

```
[0, 1, 2]
[0, 1, 2, 4]
[0, 1, 99, 2, 4]
[0, 1, 2, 4]
[0, 1, 2, 4, 5, 6, 7]
```

1.2.2 `sort` vs `sorted`

`sort` function sorts the list in place, while `sorted` returns a new sorted list.

[10]:
```
nums = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]

# returns new sorted list. Original list is not changed
print(sorted(nums))
print(nums)

# returns None. Original list is changed
print(nums.sort())
print(nums)
```

```
[1, 1, 2, 3, 3, 4, 5, 5, 5, 6, 9]
[3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]
None
[1, 1, 2, 3, 3, 4, 5, 5, 5, 6, 9]
```

1.3 Dictionary Methods

1.3.1 `keys` and `values`

[11]:
```
infoDict = {'track': 'OS', 'name': 'Ahmed', 'age': 17}
print(infoDict.keys())
print(infoDict.values())
```

```
dict_keys(['track', 'name', 'age'])
dict_values(['OS', 'Ahmed', 17])
```

1.3.2 `in` Operator

[12]:
```
print(infoDict)
print('name' in infoDict)
```

```
{'track': 'OS', 'name': 'Ahmed', 'age': 17}
True
```

1.3.3 `items` and `update`

`items` function returns a list of tuples containing the key-value pairs of the dictionary.

`update` function can do one of the following:

- Update a key's value with a new value.
- Add a new key-value pair to the dictionary.

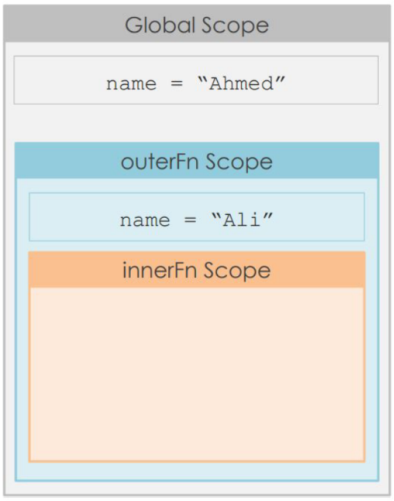
It depends on the key passed to the function, if the key is already in the dictionary, the value will be updated, otherwise a new key-value pair will be added.

As you see in the example below the track value was changed from `OS` to `JS` and since the key `branch` is not in the dictionary, it was added with the value `Benha`.

[13]:
```
print(infoDict.items())

addInfoDict = {'track': 'JS', 'branch': "Benha"}
print(f'Before: {infoDict}')
infoDict.update(addInfoDict)
print(f'After:  {infoDict}')
```

```
dict_items([('track', 'OS'), ('name', 'Ahmed'), ('age', 17)])
Before: {'track': 'OS', 'name': 'Ahmed', 'age': 17}
After:  {'track': 'JS', 'name': 'Ahmed', 'age': 17, 'branch': 'Benha'}
```

1.4 Scope



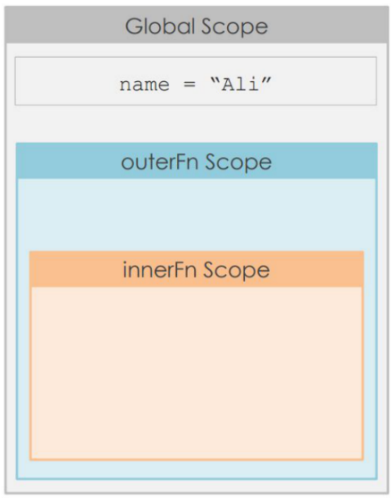
`outerFn` will create a variable `name` and assign it the value of "Ali". The variable `name` in the global scope will not be affected.

`innerFn` will print the value of the variable `name` in the outer scope, which is "Ali".

[14]:
```
name = "Ahmed"


def outerFn():
    name = "Ali"

    def innerFn():
        print(name)
    innerFn()


outerFn()
print(name)
```

```
Ali
Ahmed
```

If you use `global` keyword, the function will use the global variable instead of creating a new local variable, so changing the value of the variable will affect the global variable.



[15]:
```
name = "Ahmed"


def outerFn():
    global name
    name = "Ali"

    def innerFn():
        print(name)
```

```
    innerFn()


outerFn()
print(name)
```

```
Ali
Ali
```

1.5 Practice Questions

1.5.1 What is the output of the following code?

```
def outer_fun(a, b):
  def inner_fun(c, d):
    return c + d
  return inner_fun(a, b)

res = outer_fun(5, 10)
print(res)
```

- 15
- SyntaxError
- (5, 10)

The output will be `15`. The `outer_fun` function will call the `inner_fun` function with the arguments `5` and `10`, and the `inner_fun` function will return the sum of the two arguments.

1.5.2 What is the output of the following code?

```
def fun1(name, age=20):
  print(name, age)

fun1('Emma', 25)
```

- Emma 25
- Emma 20

The output will be `Emma 25`. The `fun1` function will print the values of the arguments passed to it, which are `'Emma'` and `25`.

1.5.3 What is the output of the following `display()` function call

```
def display(**kwargs):
  for i in kwargs:
    print(i)

display(emp="Kelly", salary=9000)
```

- `TypeError`
- `Kelly 9000`
- `('emp', 'Kelly') ('salary', 9000)`
- `emp salary`

The function output will be `emp` then `salary`. This is because the function is using the `**kwargs` parameter, which is used to pass a variable number of keyword arguments (key-value pairs) to a function.

Also since we are iterating on a key-value pair with one loop variable `i`, the output will be the keys only.

1.5.4 Select which is true for Python function

1. A Python function can return only a single value
2. A function can take an unlimited number of arguments
3. A Python function can return multiple values
4. Python function doesn't return anything unless and until you add a return statement

The second, and third options are correct because a function can take an unlimited number of arguments and can return multiple values.

The first option is obviously wrong because a function can return multiple values. The last option is wrong because **any function in Python returns `None` by default (There is no `void` functions in Python).**

1.5.5 Choose the correct function declaration of `fun1()` so that we can execute the following function call successfully

```
fun1(25, 75, 55)
fun1(10, 20)
```

- `def fun1(**kwargs)`
- No, it's not possible in Python
- `def fun1(args*)`
- `def fun1(*data)`

The correct answer is `def fun1(*data)`. This is because the function is called with multiple arguments, and the `*data` parameter is used to pass a variable number of arguments to the function.

1.5.6 Given the following function `fun1()` Please select all the correct function calls

```
def fun1(name, age):
  print(name, age)
```

1. Both `fun1("Emma", age=23)`, and `fun1(age=23, name="Emma")`
2. `fun1(name="Emma", 23)`
3. `fun1(age=23, "Emma")`

Only the first option is correct.

1.5.7 What is the output of the following `display_person()` function call

```
def display_person(*args):
  for i in args
    print(i)

display_person(name="Emma", age=25)
```

- `TypeError`
- Emma 25
- name age

The output will be `TypeError` because the function is using `*args` which is used to pass a variable number of arguments to a function, but the function call is passing keyword arguments.

1.5.8 What is the output of the following function call

```
def outer_fun(a, b):
  def inner_fun(c, d):
    return c + d

  return inner_fun(a, b)
  return a

result = outer_fun(5, 10)
print(result)
```

- 5
- 15
- (15, 5)
- `SyntaxError`

It will return `15`. The `outer_fun` function is called with the arguments `5` and `10`, and it will return the result of calling `inner_fun` with `5` and `10`, which is `15`.

1.5.9 What is the output of the `add()` function call

```
def add(a, b):
  return a+5, b+5

result = add(3, 2)
print(result)
```

- 15
- 8

- (8, 7)
- Syntax Error

The output will be `(8, 7)`. The `add` function will return a tuple containing the sum of `a` and `5`, and the sum of `b` and `5`.

> If we have a function that returns multiple values, the function will return a tuple containing the values. We can also unpack the tuple to get the values separately.

1.5.10 What is the output of the following function call

```
def fun1(num):
  return num + 25

fun1(5)
print(num)
```

- 25
- 5
- `NameError`

It will be a `NameError` because the variable `num` is defined in the function `fun1`, and it is not accessible outside the function, and returning it will just pass the value of the variable, not the variable itself.