PYTHON COURSE

ENTRY LEVEL

Basics of programming in Python 3.10

This course will cover part of the arguments found in PCEP™ – Certified Entry-Level Python Programmer Certification

OPERATORS AND OPERATIONS

OPERATORS IN PYTHON

Used to **perform operations between values** or objects, in python

- **Numeric** operators
- **String** operators
- **Assignment** and **shortcut** operators
- **Unary** and **binary** operators
- **Bitwise** operators
- **Boolean** operators
- **Relational** operators

OPERATIONS

- Operations are composed by values and operators, and produces some results. Result’s value-type depends on the operation performed.
- Different operations can be performed on different data-types
- Operation’s result can be assigned to a variable

```python
a = 35
b = 32
c = a + b
# c value will be the result of the operation performed
print(c)
```

```
[> python3 operations.py
 67
```

MUTABILITY OR IMMUTABILITY?

We can **define** the **value of an object in a certain moment of the program as its “State”**.

Mutability and Immutability concepts, are **referred to object’s states**. **Some type of objects**, like Numeric literals or String literals, **don’t allow their in-memory state to be changed after creation**, that means **they’ll maintain the same exact value even after an operation it’s performed**. **This condition is called Immutability.**

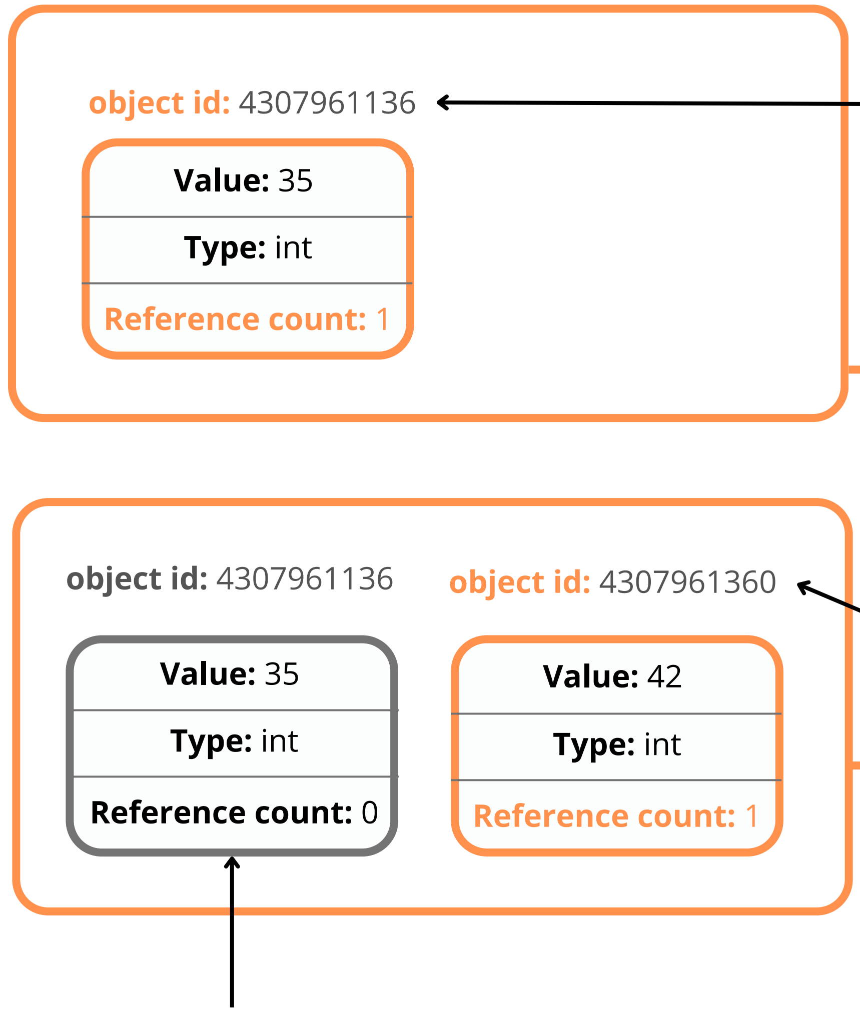
Mutable objects instead **do consent their state to be changed during the program.**

That means, when working with variables holding immutable types, you need to re-assign the new value to mantain the modification performed in memory.

PYTHON

IMMUTABILITY: HOW IT WORKS

HEAP



object id: 4307961136

Value: 35
Type: int
Reference count: 1

object id: 4307961136

Value: 35
Type: int
Reference count: 0

object id: 4307961360

Value: 42
Type: int
Reference count: 1

This object will be removed from memory by Garbage Collector cause of Reference count = 0

PROGRAM

```
# variable a stores a numeric value, which is immutable
# the object in memory is crearted, whit value 32, and reference count 1

a = 35
```

reference of a is pointing on the object created in the heap

```
# an operation is performed on a, adding 7

a + 7
```

operation will not change object's in-memory value (state) as int is immutable

```
# now we print on console the value of a
```

HEAP STATE IN THIS MOMENT

```
print(a)
# 35

print(id(a))
# 4307961136

# to hold the new value you need to overwrite the a variable (or create a new one)

a = a + 7 # it's the same as 35 + 7
```

a new object with value 35 + 7 is created, 'a' now points it

```
print(a)
# 42

print(id(a))
# 4307961360

# the reference of a is pointed to a new object
```

IMMUTABLE TYPES: EXAMPLES

String literals and Numeric literals are immutable, as a reminder:

- int
- float
- str

PYTHON

NUMERIC OPERATORS & OPERATIONS

Operations between numerical values can be performed using numeric operators:

- **Sum, Multiplication, Division, Subtraction**

```
# Variable 'a' is type int Numeric

a = 35

b = a + 7 # Sum operation, + operator (sum operator)
# Results in 35 + 7 = 42

c = a * 3 # Multiplication, * operator (multiplication operator)
# Results in 35 * 3 = 105

d = a / 5 # Division, / operator (division operator)
# Results in 35 / 5 = 7

e = a - 5 # Subtraction, - operator (subtraction operator)
# Results in 35 - 5 = 30
```

NUMERIC OPERATORS & OPERATIONS

- **Floor division:** result in integer version (floating part is approximated)

```
a = 4 // 3

# results in 1
```

- **Modulo:** result in the remainder of standard division:

```
a = 15 % 3

# a value is 0, reminder of deviding 15 / 3 (result is 5, with 0 rest)
```

NUMERIC OPERATORS & OPERATIONS

- **Power:** result is the first number elevated to the power of the second

```
a = 3 ** 2

# a value is 3 squared, 9
```

- Divisions result is always considered as float type (except for floor division)
- Numeric operations with one or more float, will result in float-type
- type changing (cast) in most of the case is automatically handled by python

OPERATIONS: PRIORITY

- Operations are by default resolved from left to right
- Mathematical priority in numerical operations is maintained by default
- Use round parentheses “(” and “)” to establish priorities in operations (can be nested)

```python
operation__result = 3 + 2 * 5
print(operation__result) # 13
operation__result = (3 + 2) * 5
print(operation__result) # 25
```

QUESTION TIME

QUESTION 1:

Fill the gaps (One correct):

_________ objects, consent their state to be changed during the program after creation.

1. Mutable
2. Immutable
3. Neither
4. Python

QUESTION 2:

Fill the gaps (Find order):

____________ are performed between _______ by using ___________, results can be stored in __________________.

1. Variables
2. Values
3. Operations
4. Operators

QUESTION 3:

____, ___ and ____ are examples of immutable types.

1. memory, heap, operations
2. keywords, comments, names
3. int, float, str
4. variables, id, references

QUESTION 4:

What is the output of the following python program:

```
a = 5
b = 10.0

sum_result = a + b
multi_operator_result = ((sum_result - 2) + 7) * 2

print(sum_result)
print(multi_operator_result)
```

1. 15.0, 40.0
2. code will produce SyntaxError at line 5
3. 15, 40
4. 15.0, 40

PYTHON

QUESTION 5:

What is the output of the following python program:

```
operation_result = 4 % 2
operation_result + 10

operation_result = 10 / operation_result

print(operation_result)
```

1. 1.0

2. 1

3. 0

4. you'll get an error trying to divide by 0 (ZeroDivisionError) at line 4

EXERCISES