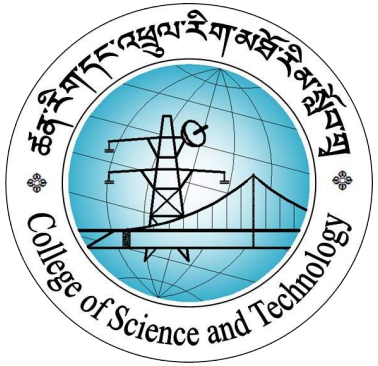


Royal University of Bhutan

Unit III:
Understanding Control Structures

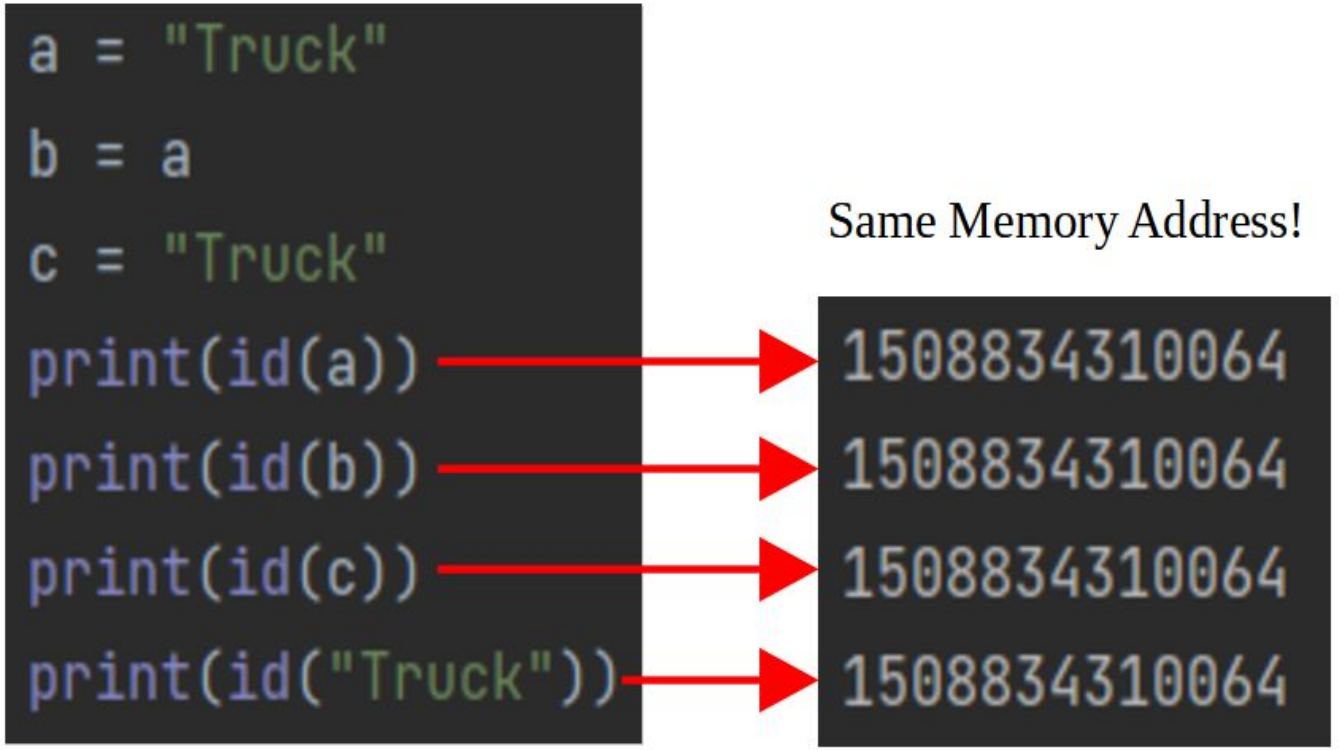
Programming Methodology (CSF101)

Outline

- Loops
- Functions, Call Stack, Scope and Function Recursion
- Memory Addresses and Pointers
- Call by Value and Call by Reference

Memory Addresses

- A reference to a specific memory location.
- The value of a variable is given a memory address. Therefore, variables having the **same value** will have the **same memory address**
- Memory address can be accessed using the built-in **id()** function



Memory Addresses

```
x = 100
print(id(x))

print(id(100))

b = 100
print(id(b))

x = [1, 2, 3]
print(id(x))
```

```
4376565200
4376565200
4376565200
4377427712
```

Memory Addresses

Python memory is divided into two:

- Stack memory : Stores temporary data specific to a function call.
- Heap Memory : Stores objects and larger data structures that require a longer lifespan than stack memory can provide.

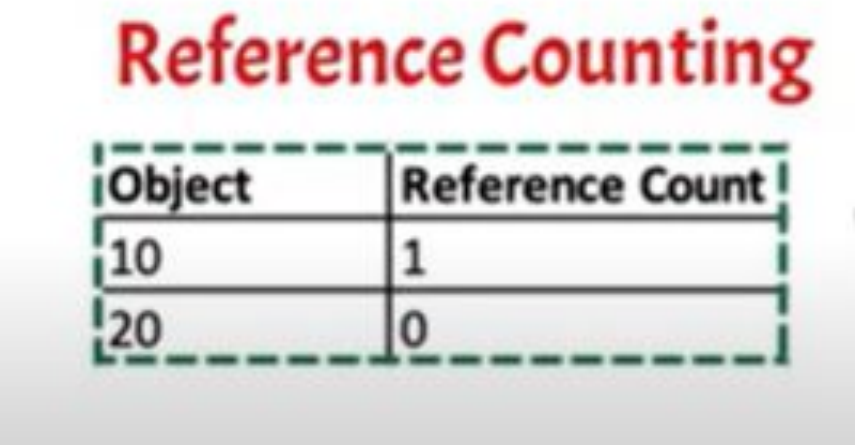
Garbage Collector : an automatic memory management technique used to reclaim memory occupied by objects that are no longer in use by a program.

Uses the concept of reference counting.

Memory Addresses

Reference Counting: Python interpreter keeps a table where it counts the number of references to each object.

If the reference count is zero, the garbage collector will then remove the object from the memory.

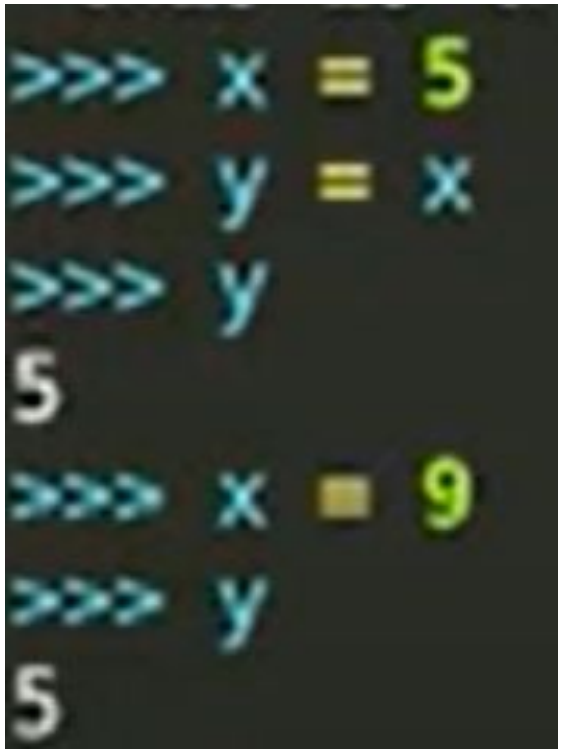


Pointers

- Variables that hold the memory address of another variable.

- In python, variables can't point to other variables but instead it points to the value of that variable.

- Therefore, python doesn't have pointers but some can be achieved using techniques that mimic pointer-like behavior.



Call by Value

A copy of the original variable is passed, so the original variable will not change.

For immutable objects like integers, floats, strings.

There are two copies of parameters stored in different memory locations. One is the original copy and the other is the function copy.

Any changes made inside functions are not reflected in the actual arguments of the caller.

Call by value

Example

```
def myFunc(a):
    print("Value received in 'a' =", a)
    a += 2
    print("Value of 'a' changes to :", a)

num = 13
print("value of number before function call: ", num)
myFunc(num)
print("Value of number after function call= ", num)
```

```
value of number before function call:  13
Value received in 'a' = 13
Value of 'a' changes to : 15
Value of number after function call=  13
```

Call by reference

The variable itself is passed, so the original variable will be altered. (mutable objects like list, dictionary and set)

The address of the arguments is passed to the function as the parameters.

Both the actual and formal parameters refer to the same locations.

Any changes made inside the function are actually reflected in the arguments of the caller.

Call by reference

```
def myFunc(myList):
  print("List received: ", myList)
  myList.append(3)  # Add 3 to the end of the list
  myList.extend([7, 1])  # Extend the list with [7, 1]
  print("List after adding some elements:", myList)

  myList.remove(7)  # Remove the first occurrence of 7
  myList[:] = [3, 4, 6]  # Reassign the list (effective
  print("List within called function:", myList)
  return

List1 = [1]
print("List before function call :", List1)
myFunc(List1)
print("List after function call: ", List1)
```

```
List before function call : [1]
List received:  [1]
List after adding some elements: [1, 3, 7, 1]
List within called function: [3, 4, 6]
List after function call:  [3, 4, 6]
```

Reference

Swain, A. (2022, July 6). What is Call by Value and Call by Reference in Python? - Scaler Topics. Scaler Topics. https://www.scaler.com/topics/call-by-value-and-call-by-reference-in-python/


THANK YOU