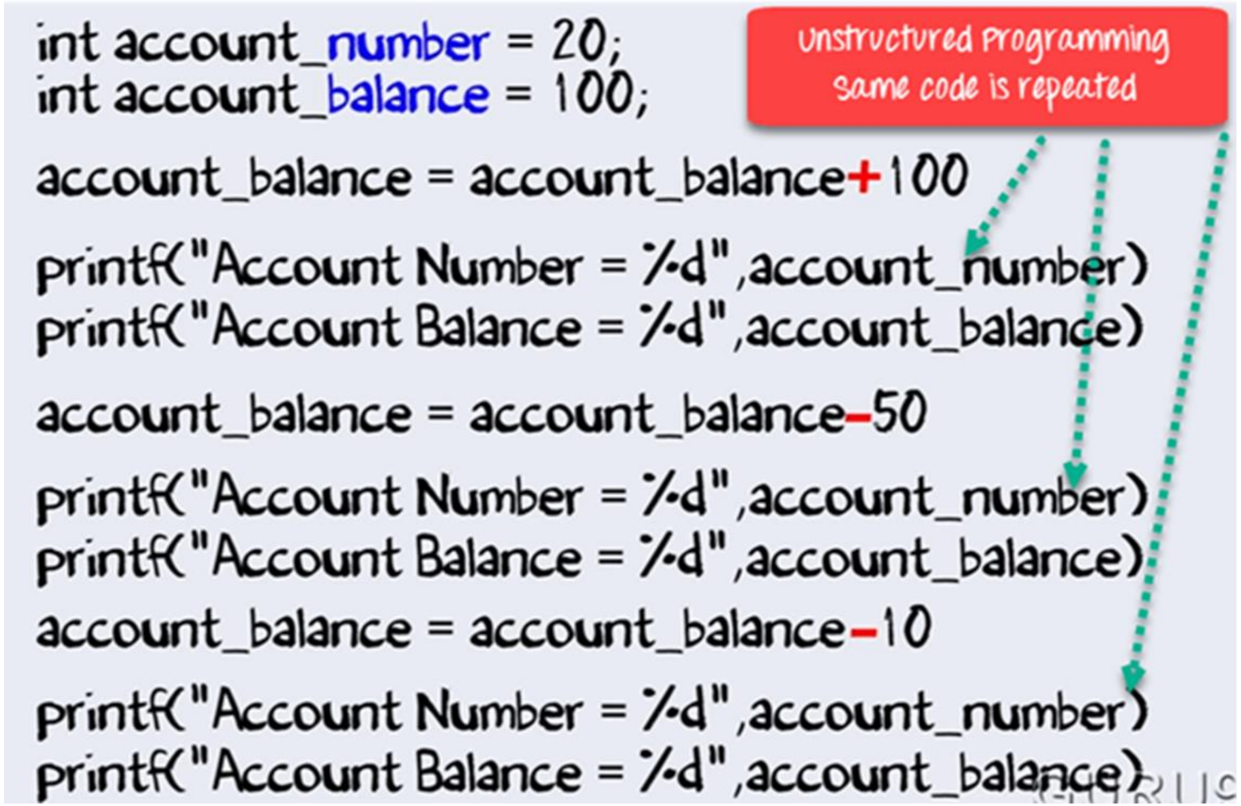
PYTHON

Lecture - 06

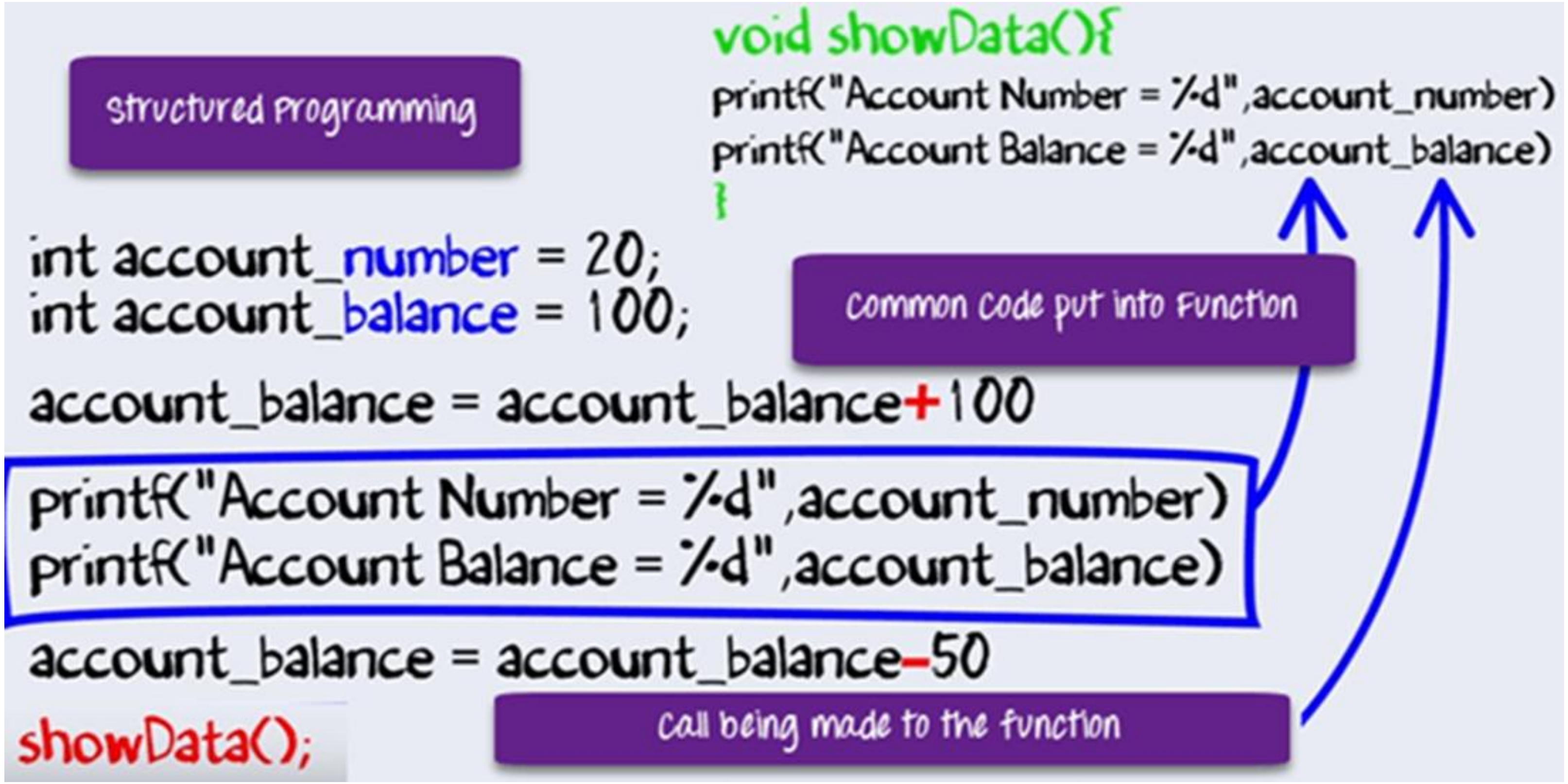
Lecture Topics

- Python Functions & Modules (Part – **01**)

Recall: C Unstructured Code Segment


```
int account_number = 20;
int account_balance = 100;

account_balance = account_balance+100

printf("Account Number = %d",account_number)
printf("Account Balance = %d",account_balance)

account_balance = account_balance-50

printf("Account Number = %d",account_number)
printf("Account Balance = %d",account_balance)
account_balance = account_balance-10

printf("Account Number = %d",account_number)
printf("Account Balance = %d",account_balance)
```

Recall: C Structured Code Segment


Structured Programming
Common Code put into Function
Call being made to the function

```
void showData(){
printf("Account Number = %d",account_number)
printf("Account Balance = %d",account_balance)
}
```

```
int account_number = 20;
int account_balance = 100;

account_balance = account_balance+100

printf("Account Number = %d",account_number)
printf("Account Balance = %d",account_balance)

account_balance = account_balance-50

showData();
```

Python Functions

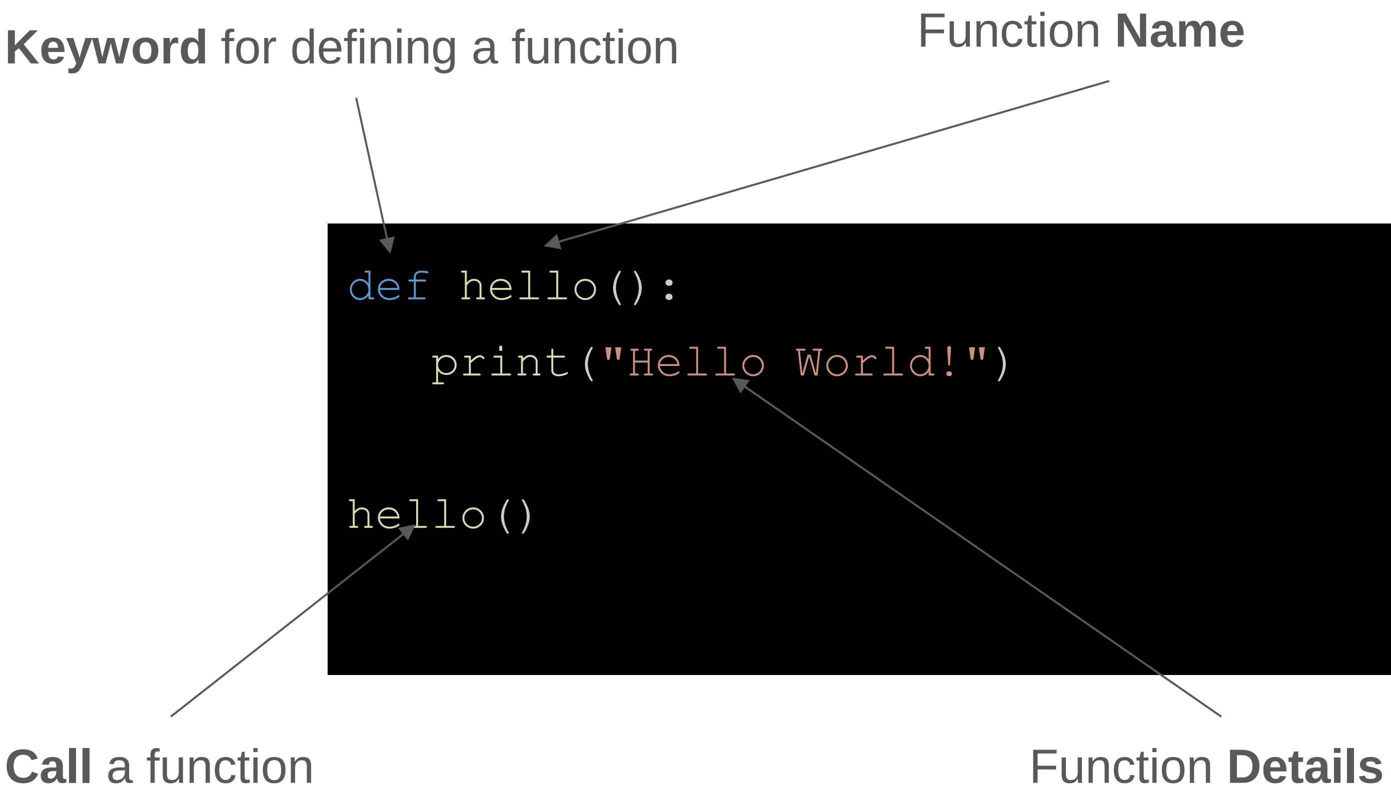
A function is like a mini-program within a program.

You’re already familiar with the **print()**, **input()**, and **len()** functions from the previous lectures. It's called python built-in function.

For better understand how python function works, let’s create one…

```
def hello():
   print("Hello World!")

hello()
```

Python Functions



```python
def hello():
    print("Hello World!")

hello()
```

Python Functions

def Statements with **Parameters**:

Parameter

```python
def hello(name):
  print("Hello " + name + "!")


hello("Joy")
hello("Rini")
```

Argument

Python Functions

def Statements with **return** type:

```
def sum(a,b):
    s = a+b
    return s

x = sum(5,6)
print(x)
```

Python Functions

def Statements with **multiple-return** type:

```
def check():
    return "ok",10, 3.14

str, num1, num2 = check()
print(str)
print(num1)
print(num2)
```

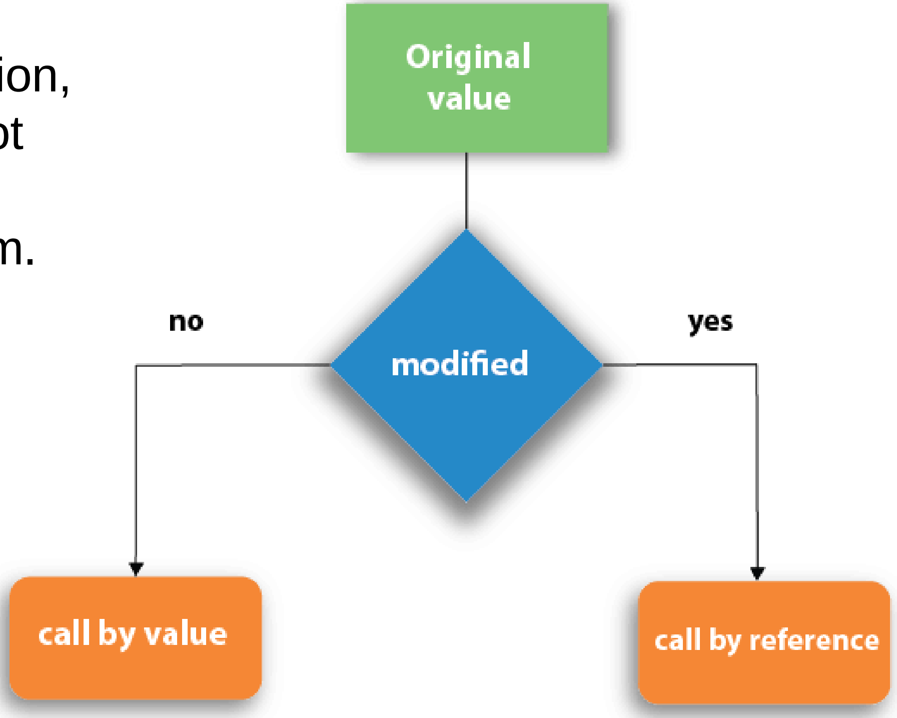
```
ok
10
3.14
```

Python Functions

Call by Value vs **Call by Reference**

When a variable is passed to a function, if any changes to its variable does not get reflected in the actual argument, then it uses **call by value** mechanism. i.e., C and C++

On the other hand, if the change is reflected, then it becomes **call by reference** mechanism. i.e., Python



Python Functions

Call by Value vs **Call by Reference**

pass by reference
pass by value
cup =
fillCup( )
cup =
fillCup( )

Python Functions

Python use **call by reference** technique. → How to prove?

Can use **id()** → it return memory location for a variable or object.

```
def testfunction(arg):
  print ("ID inside the function:", id(arg))

var="Hello"
print ("ID before passing:", id(var))
testfunction(var)
```

Python Functions

```python
def testfunction(arg):
  print ("ID inside the function:", id(arg))

var="Hello"
print ("ID before passing:", id(var))
testfunction(var)
```

```
ID before passing: 137039164252016
ID inside the function: 137039164252016

# memory address can be vary based on device
# but it remains same before and after function call.
```

Python Functions

Python uses **call by reference** technique. **BUT…**

The behaviour also depends on whether the passed object is **mutable or immutable**.

→ **Mutable objects** can be changed after they are created. i.e. lists, dictionaries.

→ **Immutable objects** cannot be changed after they are created, it creates new object when it performs operation. i.e. strings, integers, or tuples.

Python Functions

Python uses **call by reference** for mutable objects.

But immutable object such as numeric object. **This works as follows:**

When a numeric object is passed, and then the function changes the value, it actually **creates a new object** in the memory, **leaving the original variable unchanged**.

Python Functions

```
def testfunction(arg):
  print ("ID inside the function:", id(arg))
  arg=arg+1
  print ("new object after increment:", arg, id(arg))

var=10
print ("ID before passing:", id(var))
testfunction(var)
print ("value after function call:", var)
```

```
ID before passing: 140719550297160
ID inside the function: 140719550297160
new object after increment: 11 140719550297192
value after function call: 10
```

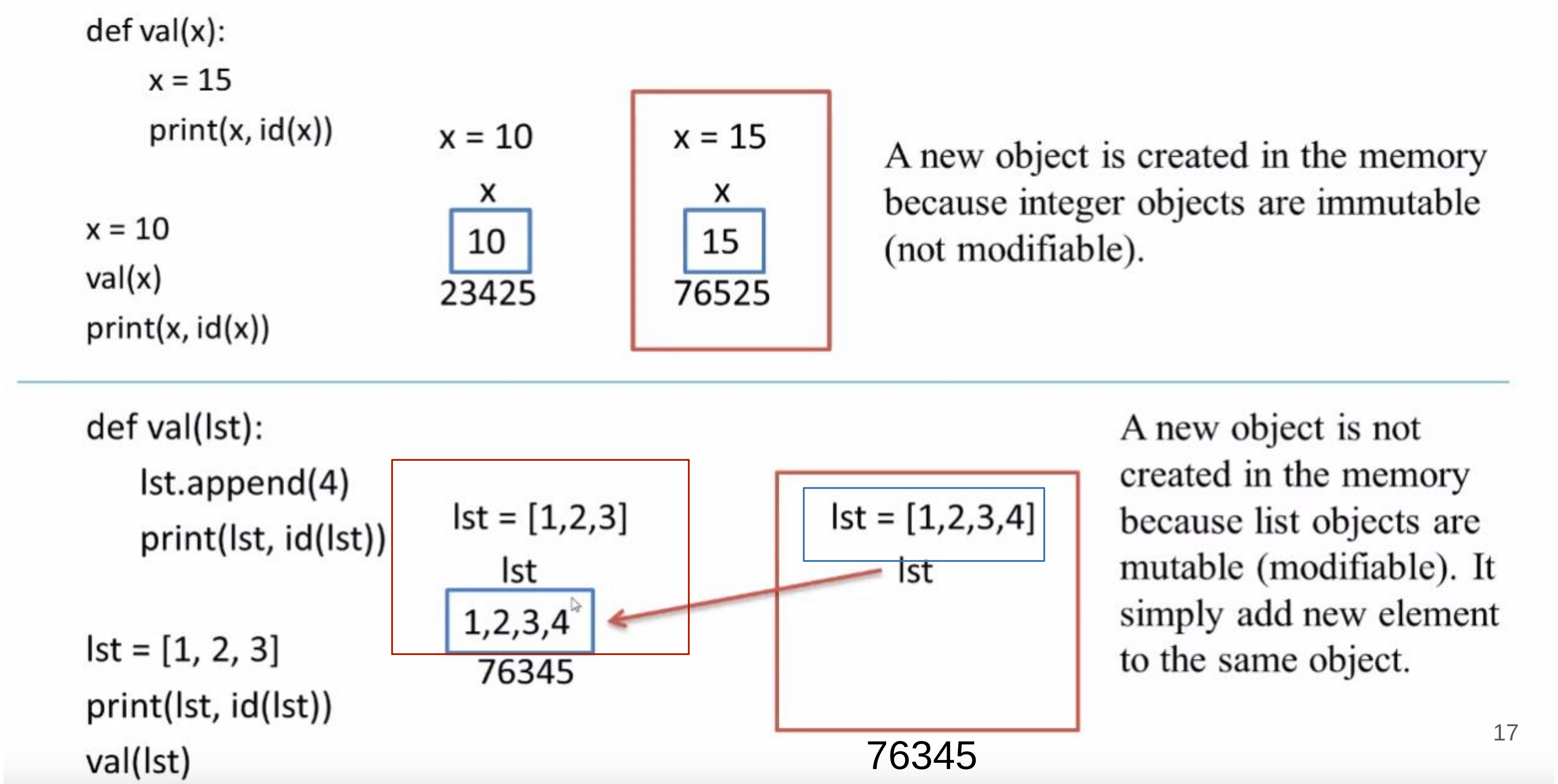
Python Functions

Python uses **call by reference** technique. **AND…**

Python **List** object is also **mutable**.

It is also passed by **reference**, as the **id()** of list before and after passing is same. However, if we modify the list inside the function, its global representation also **reflects the change**.

Python Functions



```
def val(x):
    x = 15
    print(x, id(x))

x = 10
val(x)
print(x, id(x))
```

A new object is created in the memory because integer objects are immutable (not modifiable).

```
def val(lst):
    lst.append(4)
    print(lst, id(lst))

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

A new object is not created in the memory because list objects are mutable (modifiable). It simply add new element to the same object.

Python Functions

```
def testfunction(arg):
  print ("Inside function:",arg)
  print ("ID inside the function:", id(arg))
  arg=arg.append(100)

var=[10, 20, 30, 40]
print ("ID before passing:", id(var))
testfunction(var)
print ("list after function call:", var)
```

```
ID before passing: 132562306735872
Inside function: [10, 20, 30, 40]
ID inside the function: 132562306735872
list after function call: [10, 20, 30, 40, 100]
```

Python Functions

Positional or Required Arguments

```
def printme( str ):
   print (str)
   return
printme()
```

As we pass parameter here

We must pass a argument here

What is the **output** of the code**???**

```
Traceback (most recent call last):
  File "/home/ribnat/Python.py", line 5, in <module>
    printme()
TypeError: printme() missing 1 required positional argument: 'str'
```

WHY ??

Python Functions

Keyword Arguments

We can pass arguments in different order.

```
def printinfo( name, age ):
  print ("Name: ", name)
  print ("Age: ", age)
  return

printinfo( age=50, name="miki" )
```

```
Name:  miki
Age:  50
```

Python Functions

Default Arguments

A default argument is an argument that assumes a default value if a value is not provided in the function call for that argument.

```
def printinfo( name, age = 35 ):
  print ("Name: ", name)
  print ("Age: ", age)
  return

printinfo( name="miki" )
```

```
Name:  miki
Age:  35
```

As we don't pass **age** arg, it takes **35** as default.

Python Functions

Default Arguments

A default argument is an argument that assumes a default value if a value is not provided in the function call for that argument.

```
def printinfo( name, age = 35 ):
  print ("Name: ", name)
  print ("Age: ", age)
  return

printinfo( "miki", 50 )
```

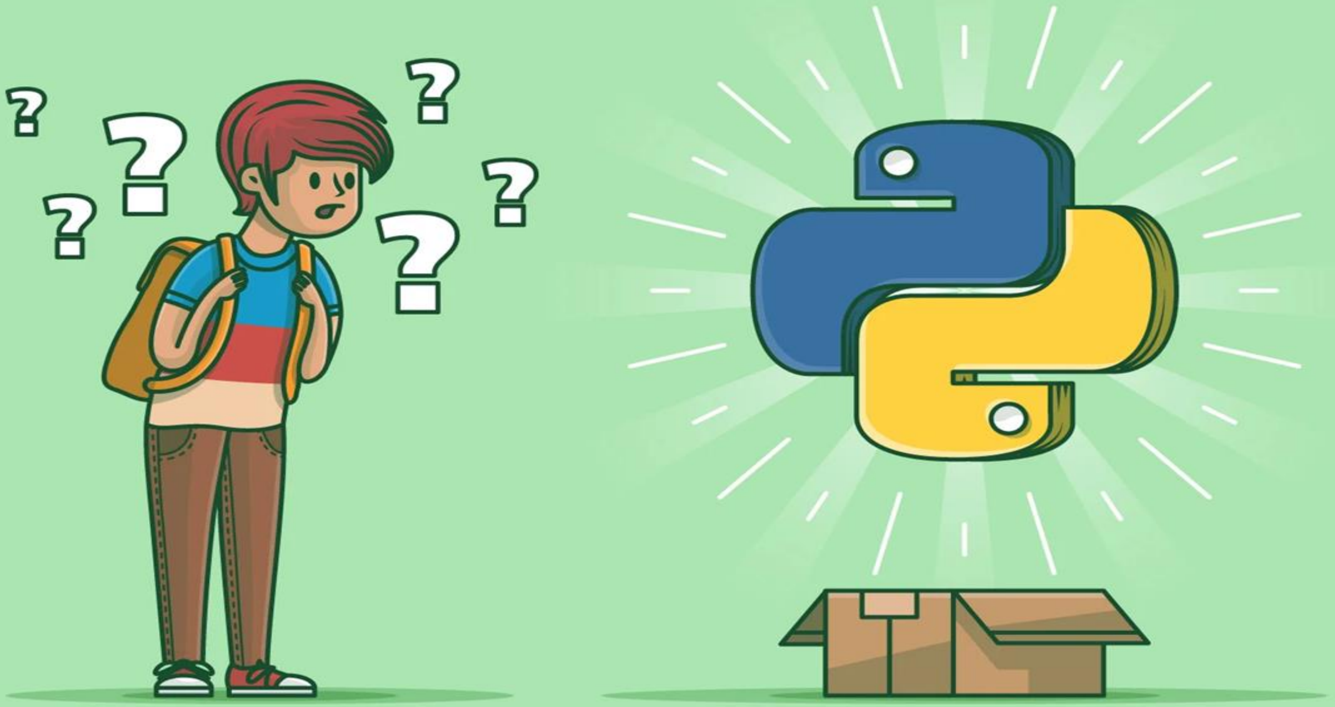
```
Name:  miki
Age:  50
```

Python Functions

Built-in Mathematical Functions

abs()	Returns the absolute value of x, i.e. the positive distance between x and zero.	X = abs(-5) → 5
max()	Returns the largest of its arguments or largest number from the iterable (list or tuple).	max(5,9) →9
min()	Returns the smallest of its arguments or smallest number from the iterable (list or tuple)	min(5,9) →5
pow()	Returns x raised to y. It is equivalent to x**y. The function has third optional argument mod. If given, it returns (x**y) % mod value.	pow(2,3) →8
round()	Returns x rounded to n digits from the decimal point.	round(5.8) →6
sum()	Returns the sum of all numeric items in any iterable (list or tuple). An optional start argument is 0 by default. If given, the numbers in the list are added to start value.	sum(5,6) →11

Exercise Time



Exercise – 6.1

1. Print all even numbers from **1** to **N**.

2. Print all numbers from **1** to **N** **except** those, that are divisible by **5**.

3. Print all numbers from **1** to **N** that are divisible by **3** or **4** but not **5**.

4. Print all prime numbers from **1** to **N**.

5. Given **N** numbers, find the **largest** number.

Exercise – 6.2

Write a Python program that computes the greatest common divisor (GCD) of two positive integers.

Input:
8
12

Output:
4

Exercise – 6.2 (ans)

```
x = 86
y = 44
n = min(x,y)
gcd = 0
for i in range(1,n+1):
   if x%i == 0 and y%i == 0:
       gcd = i

print(f"The GCD of {x} and {y} is {gcd}.")
```

# It's called formatted string, which helped to print
# variables with strings.

Exercise – 6.3

Write a Python program to find the least common multiple (LCM) of two positive integers.

Input:
```
8
12
```

Output:
```
24
```

Exercise – 6.3 (ans)

```python
num1 = 8
num2 = 12
for i in range(max(num1, num2), 1 + (num1 * num2), max(num1, num2)):
   if i % num1 == i % num2 == 0:
       lcm = i
       break

print("LCM of", num1, "and", num2, "is", lcm)
```

Exercise – 6.4

Take a List as input and find its MEX.
MEX is smallest non-negative number that isn't present in the list.

Input_1: [2, 1, 4, 0, 5]
Output_1: 3

Input_2: [2, 1, 4, 0, 5, 3]
Output_2: 6

Hints: you can sort the List using the function sort(), then find out the mex using a loop, but how? Think yourself.

```
A = [5, 2, 4]
A.sort()
print(A) → [2, 4, 5]
```

Exercise – 6.4 (ans)

```python
arr = [2, 1, 4, 0, 5, 1]
N = len(arr)

arr.sort()
mex = 0
for idx in range(N):
    if arr[idx] == mex:
        mex += 1

print(mex)
```

Exercise – 6.5

You are given a phone number as a string. Report as **“INV”** if the number is invalid. Otherwise find out its operator company name. Every valid number has exact 11 digits and first 3 digits decide it’s operator company name. Here is the list of valid operators & their codes:

Grameenphone → ‘017’ or ‘013’

Teletalk → ‘015’

Banglalink →’014’ or ‘019’

Airtel →’016’

Robi →’018’

Sample Input: "01411111111"
Sample Output: “Banglalink”

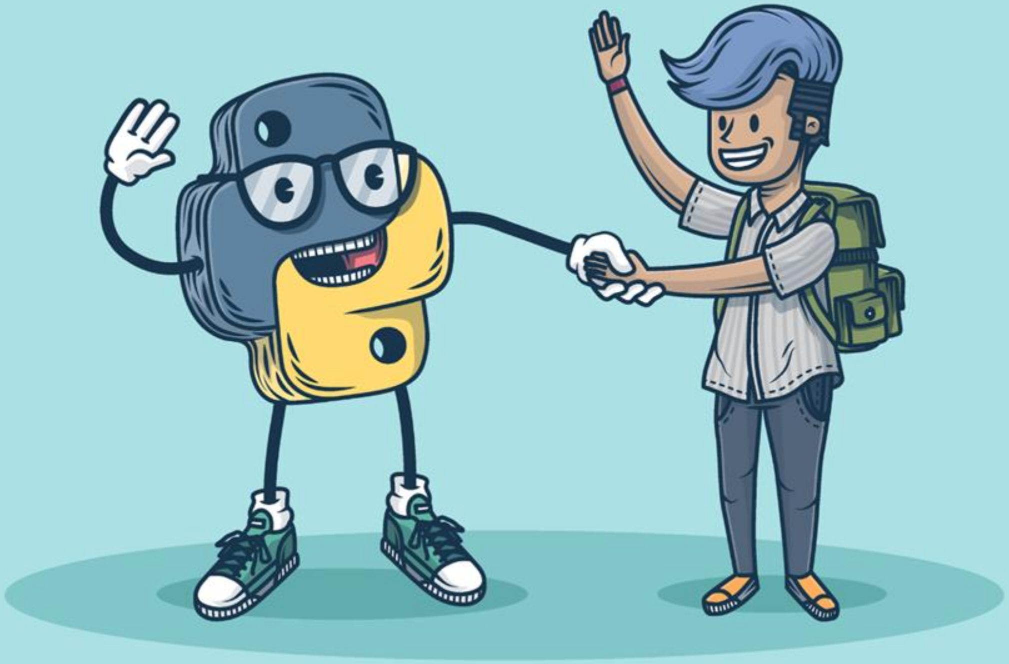
Solve the Problem using match case

Exercise – 6.5 (ans)

```python
num = input()
if(len(num)!=11):
  print("INV")

match num[:3]:
    case "017":
        print("Grameenphone")
    case "013":
        print("Grameenphone")
    case "014":
        print("Banglalink")
    case "019":
        print("Banglalink")
    case "015":
        print("Teletalk")
    case "016":
        print("Airtel")
    case "018":
        print("Robi")
    case _:
        print("INV")
```

Resources

- https://www.tutorialspoint.com/python/index.htm
- https://www.w3resource.com/python/python-tutorial.php
- https://www.w3resource.com/python-exercises/string/
- https://www.w3schools.com/python/
- https://www.geeksforgeeks.org/python-programming-language/
- https://youtu.be/t2_Q2BRzeEE?si=OO6J_YNCZykedqsT
- https://realpython.com/
- Head First Python, 3rd Edition by Paul Barry
- Automate the Boring Stuff with Python By Al Sweigart.



Thank You