



# Workshop on Python (Day 2)

By Suriya G  
Organized by Suresh Sir, UPNM



# TABLE OF CONTENTS

## 01

### Recap

- Previous day recap

## 02

### If-else

- Exploring if-else-elif






## 03

### Loops

- For
- While
- Nested Loops

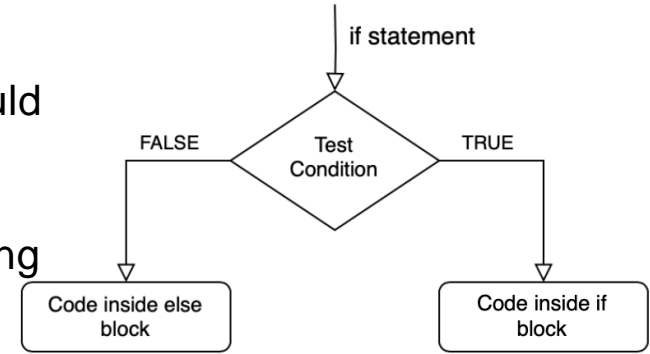
## 04

### Functions

- Types of functions
  - Parameter settings
  - Recursive cases
- 
- 
- 


# If-Else:

- There are some situations in our real life, where we should take some decisions.
- Same problem solving skills can be implemented in coding to solve technical problems.







# If:

- Executes a block of code **only if** the given condition is **True**
  - Used for decision-making in programs.
- 

## If.py

```
1  # This is to check the output of if conditions
2  if (10>5) :
3      print("10 is greater than 5")
4
```



# Elif:

- Checks **multiple conditions** when the if condition is **False**.
- Can have multiple elif blocks, but they execute sequentially until one is True.

## elif.py

```
1  # This is to check the output of elif conditions
2  num = 0
3  if num>0:
4      print("Positive number")
5  elif num==0:
6      print("Number is 0")
```






# Else:

- Executes a block of code when **all previous conditions are False**.
- Acts as a fallback condition in an if-elif-else chain.






## elif.py

```
1  # This is to check the output of else conditions
2  age = 16
3  if age>=18:
4      print("Adult")
5  elif age>=60:
6      print("Senior Citizen")
7  else:
8      print("Minor Child")
```





# Loops in python:

- A loop is used to **repeat** an instruction **multiple times** until a condition goes wrong.
  - Acts as a fallback condition in an if-elif-else chain.
- 
- 
- 
- 
- 

Consider, I want to print "UPNM" 10 times **(Method – 1) Without loops**

## print.py

```
1  # This is to print UPNM 10 times
2  print("UPNM")
3  print("UPNM")
4  print("UPNM")
5  print("UPNM")
6  print("UPNM")
7  print("UPNM")
8  print("UPNM")
9  print("UPNM")
10 print("UPNM")
11 print("UPNM")
```



Consider, I want to print “UPNM” 10 times **(Method – 2) Without Loops**

### print.py

```
1 # This is to print UPNM 10 times
2 print(f"UPNM\n" * 10)
```

Consider, I want to print “UPNM” 10 times **(Method – 3) Using loops**

### print.py

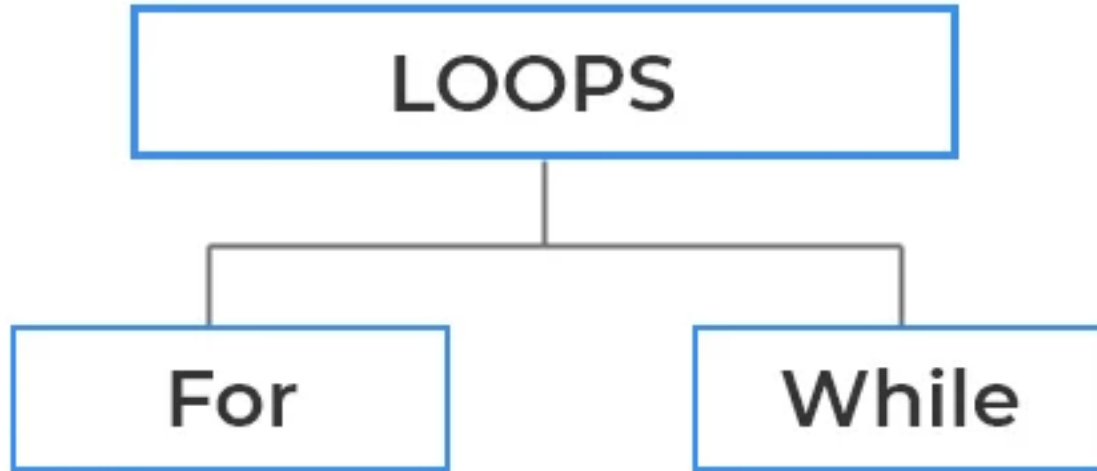
```
1 # This is to print UPNM 10 times
2 for i in range(11):
3     print("UPNM")
```

Consider, I want to print “UPNM” 10 times **(Method – 4) Using loops**

### print.py

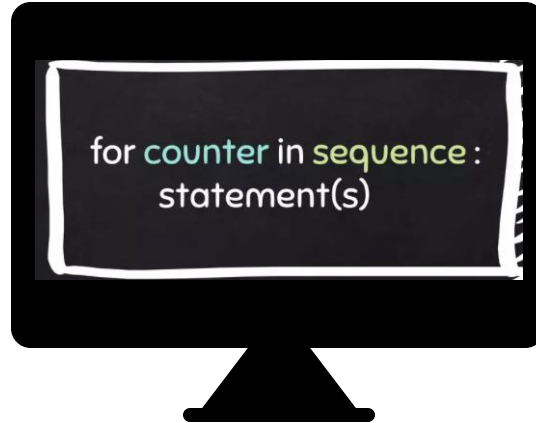
```
1  # This is to print UPNM 10 times
2  i = 1
3  while i != 11:
    print("UPNM")
    i+=1
```

# LOOPS IN PYTHON




# For Loop:

- For loop is used for iterating over a sequence.
- The sequence can be either a list, a tuple, a dictionary, a set, or a string








# For Loop (Use cases)

- Checking attendance in a classroom.
  - Sending bulk e-mails.
  - Ordering items from shopee cart.
- 

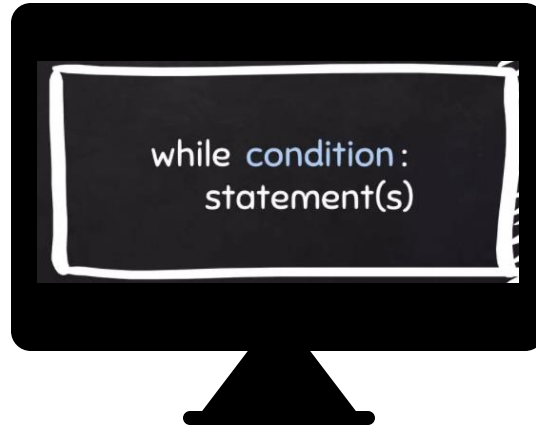
**for.py**

```
1 # This is to practice for loop
2 for i in range(70):
3     print(i)
```




# While loop:

- While loop is used to execute a set of statements as long as a **condition is true**
- Useful when the number of iterations is **unknown beforehand**.






# While Loop (Use cases)

- Game loops until you win
  - Real time sensor monitoring
  - Elevator systems in shopping mall and airport
- 

## while.py

```
1  # This is to practice while loop
2  i = 0
3  while i < 71:
4      print(i)
5      i+=7
```





# Nested Loops in Python:

- A loop inside a loop is known as a nested loop.
- The "inner loop" will be executed one time for each iteration of the "outer loop"

## Nested For loop

```
for i in range(1, 11):  
    for j in range(1, 11):  
        print(i*j, end=" ")  
    print('')
```

Diagram illustrating the execution of a nested for loop:

- The **Outer Loop** (indicated by a green bracket) is the `for i in range(1, 11):` loop.
- The **Inner loop** (indicated by a red bracket) is the `for j in range(1, 11):` loop, which is nested inside the outer loop.
- The **Body of inner loop** (indicated by a purple bracket) is the `print(i*j, end=" ")` statement.
- The **Body of Outer loop** (indicated by a purple bracket) is the `print('')` statement, which is executed after the inner loop completes its iterations.




# Break

- Immediately **exits the loop** when a condition is met.
  - Used to stop iteration prematurely.
- 


## break.py

```
1  # This is to practice break
2  for i in range(5):
3      if i==3:
4          break
5      print(i)
```






# Continue

- **Skips the current iteration** and moves to the next one.
  - Used when certain conditions require skipping logic.
- 

## break.py

```
1 # This is to practice continue
2 for i in range(5):
3     if i==3:
4         continue
5     print(i)
```



# Functions:

- Functions are sub-programs which performs tasks which may need to be repeated.
- Functions are defined using keywords, can take parameters, and can return values.

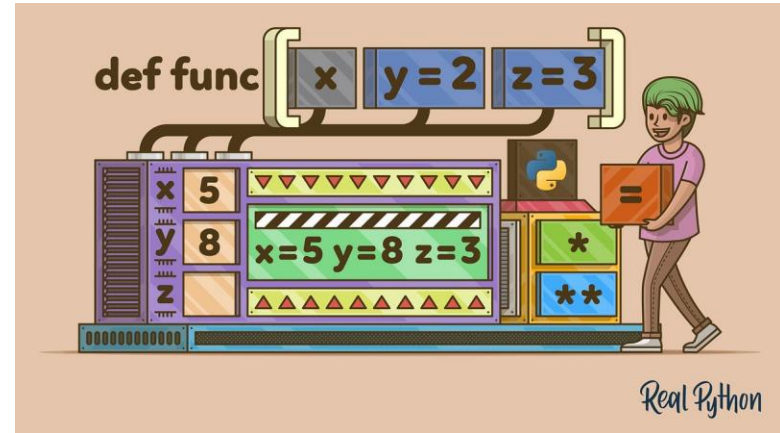


The diagram illustrates the execution flow of a Python function. It shows three lines of code: a function definition, a comment, and a function call. A blue line connects the function call to the function definition, indicating the call process. Numbered arrows indicate the flow: 1 points to the function definition, 2 points to the function call, and 3 points to the code after the function call.

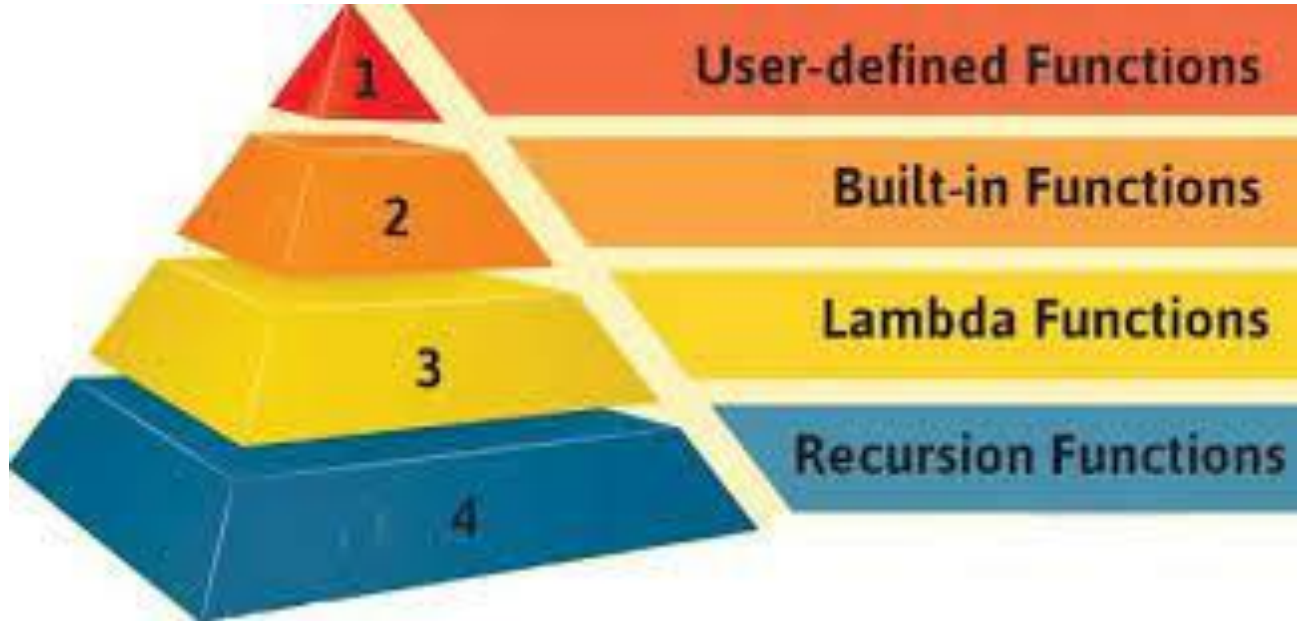
```
def greet():  
    print('Hello World!')  
  
# call the function  
greet()  
  
print('Outside function')
```

# Why do we use Functions?

- Reusability of codes.
- Reduce complexity of codes.
- Easy to maintain and understand.



# Types of Functions




# Built-in Functions

Built-in Functions in Python						
abs()	classmethod()	filter()	id()	max()	property()	str()
all()	compile()	float()	input()	memoryview()	range()	sum()
any()	complex()	format()	int()	min()	repr()	super()
ascii()	delattr()	frozenset()	isinstance()	next()	reversed()	tuple()
bin()	dict()	getattr()	issubclass()	object()	round()	type()
bool()	dir()	globals()	iter()	oct()	set()	vars()
bytearray()	divmod()	hasattr()	len()	open()	setattr()	zip()
bytes()	enumerate()	hash()	list()	ord()	slice()	__import__()
callable()	eval()	help()	locals()	pow()	sorted()	
chr()	exec()	hex()	map()	print()	staticmethod()	





# User-defined Functions

- Functions created by users for specific tasks.
  - They help us to derive custom functions using **def** keyword.
- 

## User\_function.py

```
1  # This is to practice functions
2  def square(x):
3      return x*x
4
5
```








# Return Statement:

- It will end the execution of the function.
- 


## User\_function.py

```
1  # This is to practice functions
2  def fun():
3      print("This is before return")
4      return
5      print("This is after return")
```








# Lambda Functions

- A **one-line Function** defined without name
  - It is also called as anonymous function.
- 

## lambda\_function.py

```
1  # This is to practice functions
2  square = lambda x: x*x
3  print(square)
```





# Recursive Functions:

- A function calls itself directly or indirectly.

## recursive.py

```
1  # This is to practice functions
2  def fact(n):
3      if n == 1:
4          return 1
5      else:
6          return n * fact(n-1)
```

