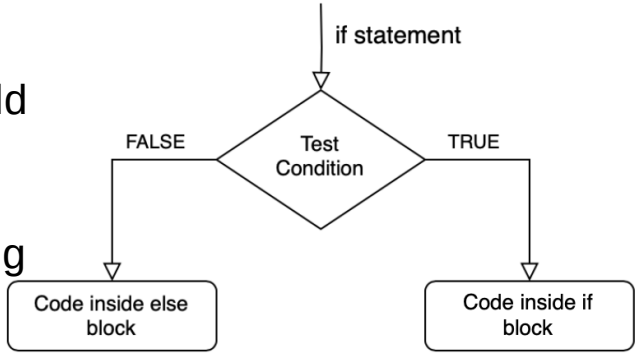
Workshop on Python (Day 2)

By Suriya G
Organized by Suresh Sir, UPNM

TABLE OF CONTENTS

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

```
# This is to check the output of if conditions
if(10>5):
    print(“10 is greater than 5”)

```

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

```
# This is to check the output of elif conditions
num = 0
if num>0:
    print(”Positive number”)
elif num==0:
    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

```
# This is to check the output of else conditions
age = 16
if age>=18:
    print(”Adult”)
elif age>=60:
    print(“Senior Citizen”)
else:
    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

```
# This is to print UPNM 10 times
print(“UPNM”)
print(“UPNM”)
print(“UPNM”)
print(“UPNM”)
print(“UPNM”)
print(“UPNM”)
print(“UPNM”)
print(“UPNM”)
print(“UPNM”)
print(“UPNM”)
```

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

print.py

```
# This is to print UPNM 10 times
print(f“UPNM\n” * 10)
```

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

print.py

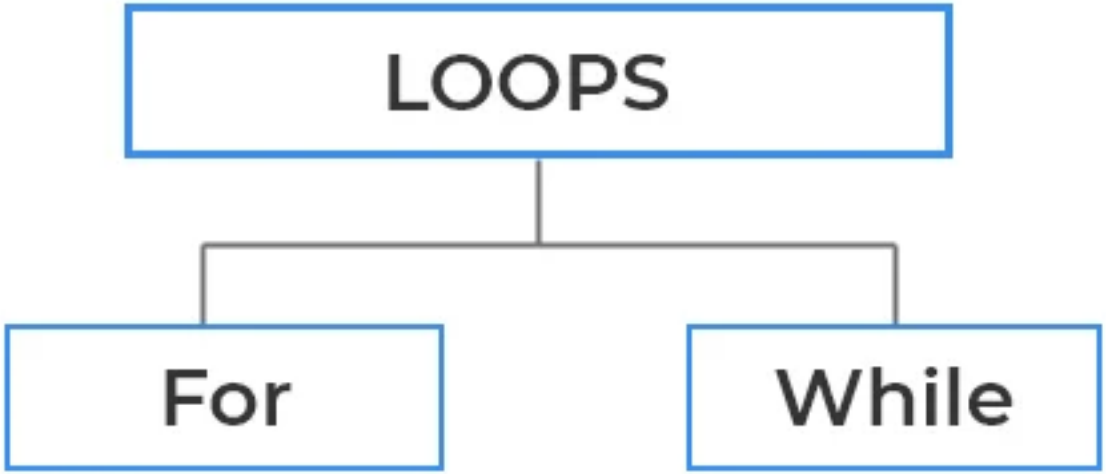
```
# This is to print UPNM 10 times
for i in range(11):
    print(“UPNM”)
```

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

print.py

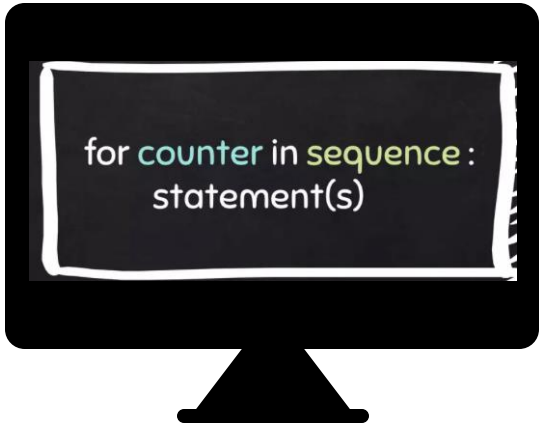
```
# This is to print UPNM 10 times
i = 1
while i != 11:
    print(“UPNM”)
    i+=1
```

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)

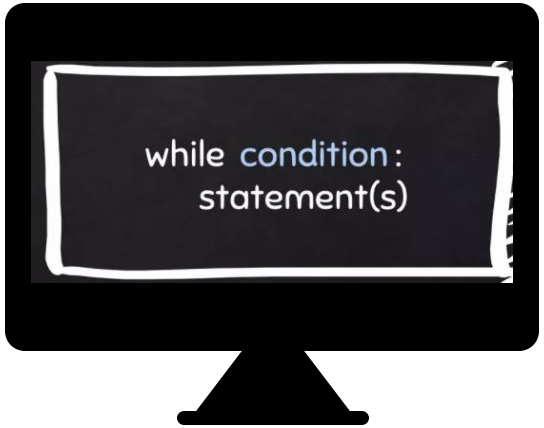
- Checking attendance in a classroom.
- Sending bulk e-mails.
- Ordering items from shopee cart.

for.py

```
# This is to practice for loop
for i in range(70):
    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)

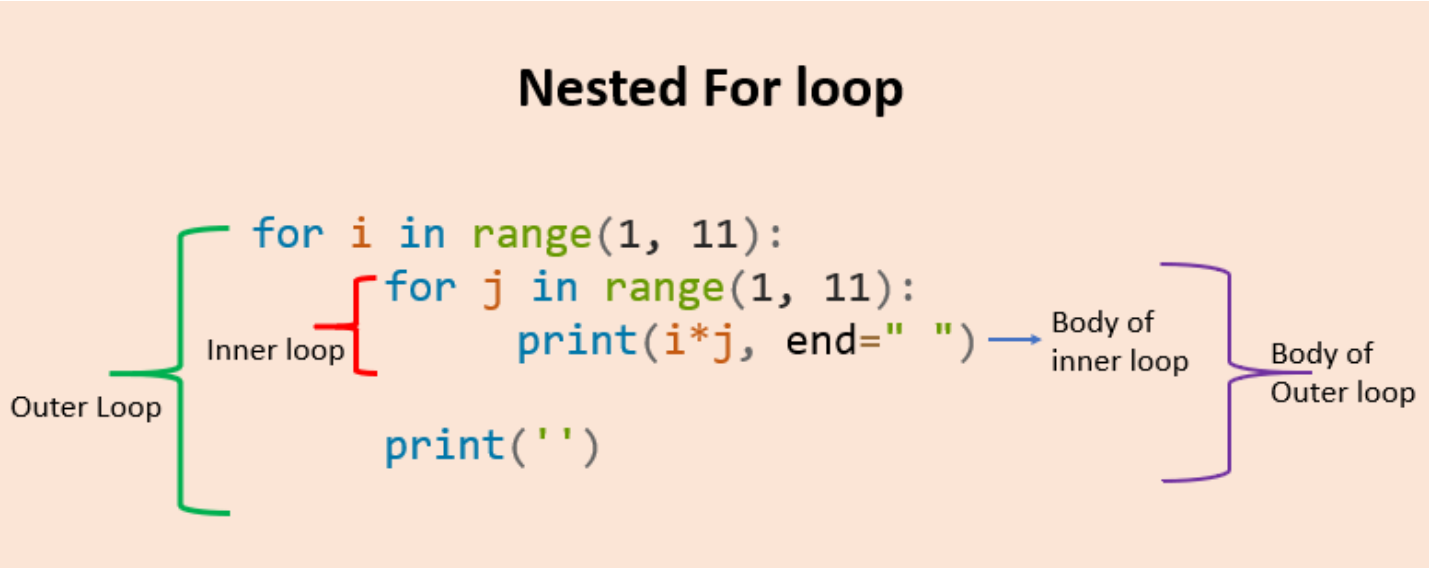
- Game loops until you win
- Real time sensor monitoring
- Elevator systems in shopping mall and airport

while.py

```
# This is to practice while loop
i = 0
while i < 71:
    print(i)
    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”



Break

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

break.py

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

Continue

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

break.py

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

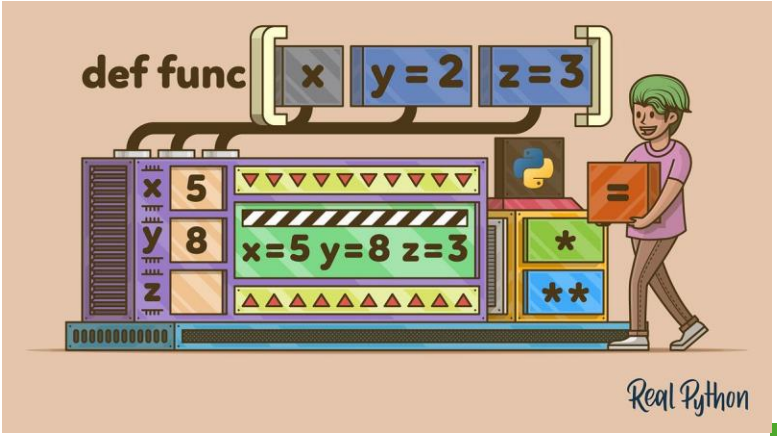
Functions:

- 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.

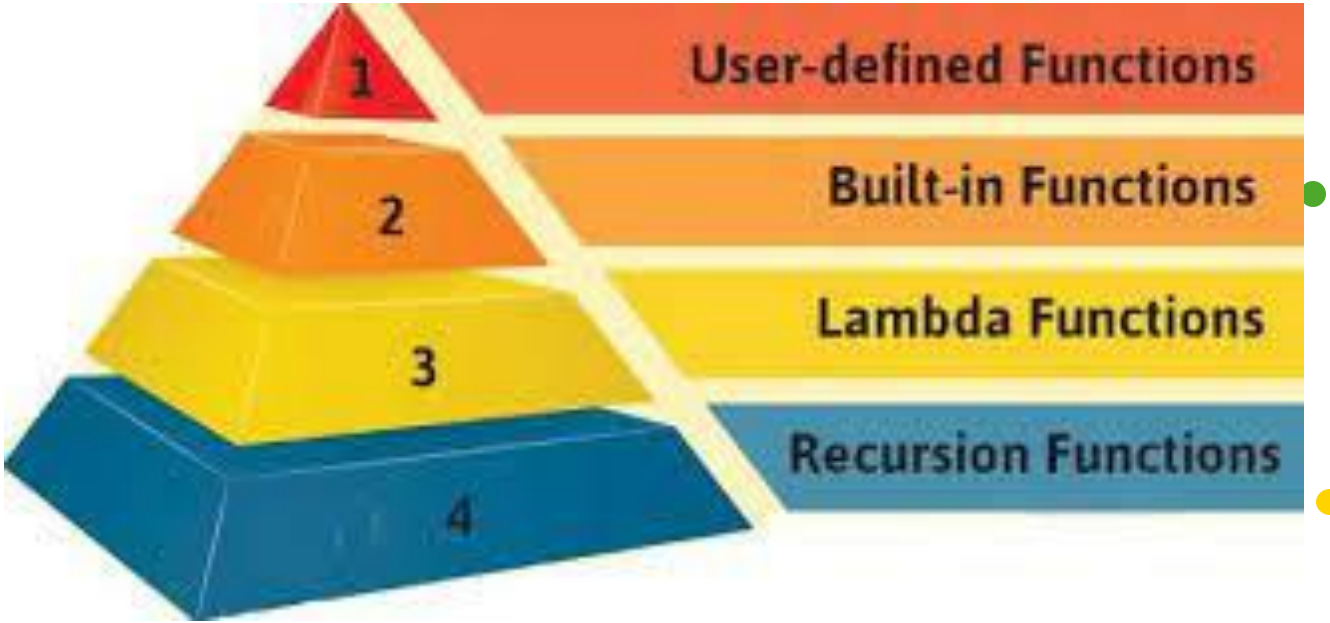
```
def greet():
  print('Hello World!')

# 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()	exce()	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

```
# This is to practice functions
def square(x):
    return x*x


```

Return Statement:

- It will end the execution of the function.

User_function.py

```
# This is to practice functions
def fun():
    print("This is before return")
    return
    print("This is after return")
```

Lambda Functions

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

lambda_function.py

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

Recursive Functions:

- A function calls itself directly or indirectly.

recursive.py

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