3. Control Structures
Functions

Subject : Programming with Python (IT3008)

Faculty: Ms. Twinkle Kosambia
Assistant Professor, Computer Science Department,
Asha M. Tarsadia Institute of Computer Science

Website: utu.ac.in

Syllabus of Programming with Python :
https://app.utu.ac.in/utuformaccess/utusyllabus.aspx?CF=7&CM=177&SY=1

Programming with Python

Subject code : IT3008

Credits : 5

Theory marks : 60+ 40= 100

Practical marks : 100 (CIE)

Reference books

Text book:

1. Allex Martelli, Anna Ravenscroft and Steve Holden, “Python in Nutshell”, 3rd Edition, O’Reilly Publication.

Reference books:

1. Magnus Lie Hetland, "Beginning Python From Novice to Professional", Third Edition, Apress, 2017.

2. David Beazley, Brian K. Jones, "Python Cookbook", 3rd edition, O’Reilly Publication, 2016.

3. Brett Slatkin, "Effective Python: 59 Specific Ways to Write Better Python", Novatec, 2016.

4. Mark Lutz "Learning Python", 4th Edition, O’Reilly Publication, 2016.

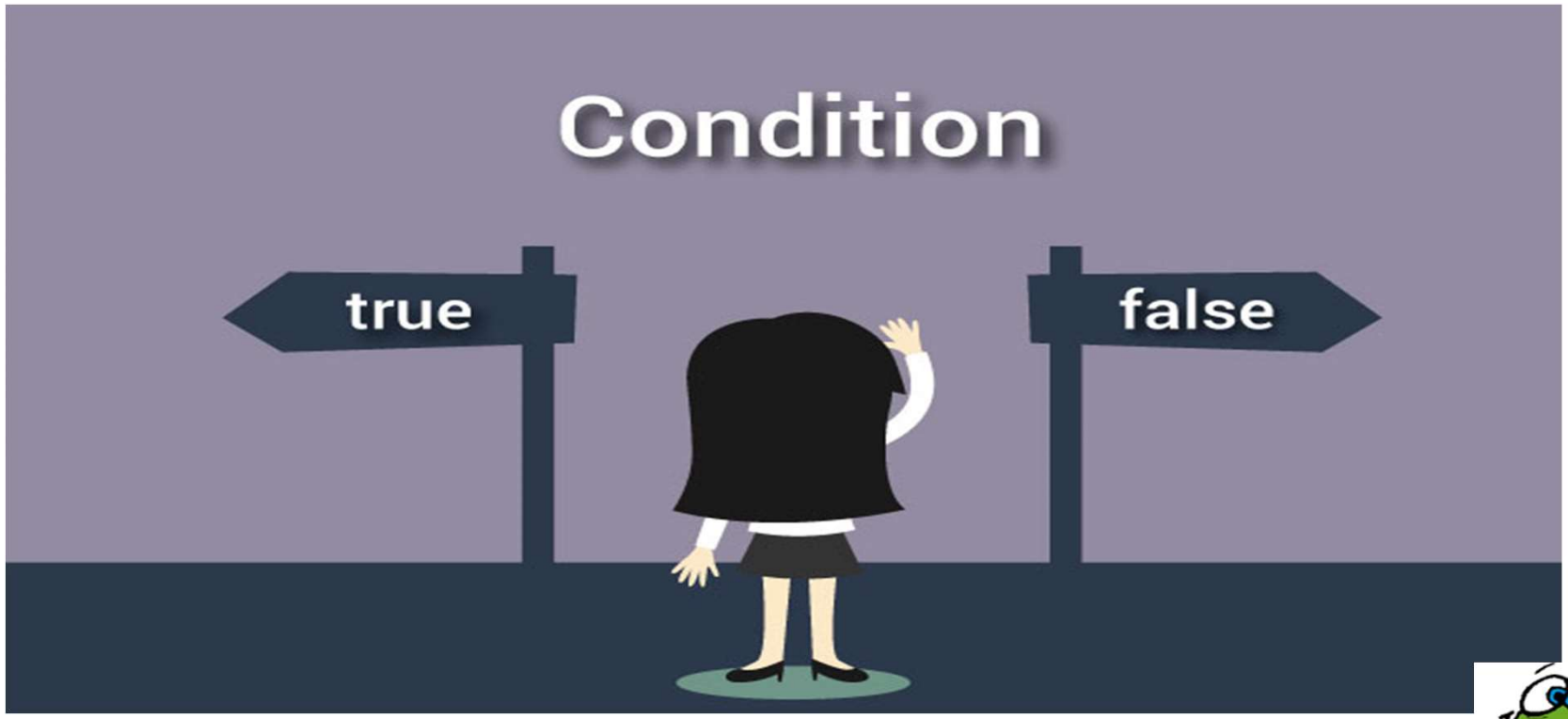
Course outcomes

Unit No.	Unit Name	Course Outcomes					
		CO1	CO2	CO3	CO4	CO5	CO6
1	Introduction to Python	✓					
2	Data Structures		✓				
3	Control Structure and Functions			✓			
4	Object Oriented Programming				✓		
5	Exception Handling and Regular Expression					✓	
6	File Handling						✓

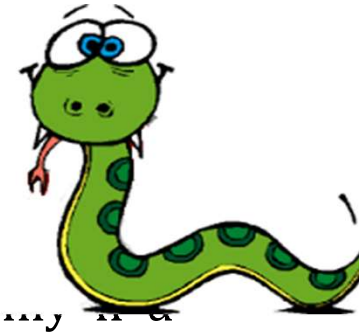
Unit – II Control Structures
Functions

Control Structures Conditional branching: if Statements, break and continue Statements, and else Clauses on loops, pass Statements, Loops: while Loops, for Loops.
Functions Defining functions, More on Defining Functions: Default argument values, Keyword arguments, Arbitrary argument lists, Unpacking argument lists, lambda Expressions, Documentation strings, Function annotations. Executing modules as scripts, The module search path, Compiled Python files, Packages: Importing * from a package, Intra-package references, Packages in multiple directories.

Python if...else Statement



Python if...else Statement



- Decision making is required when we want to execute a code only if a certain condition is satisfied.
- The if.. elif ...else statements is used in Python for decision.

Python if Statement Syntax:

```
if  test expression:
        statement(s)
```

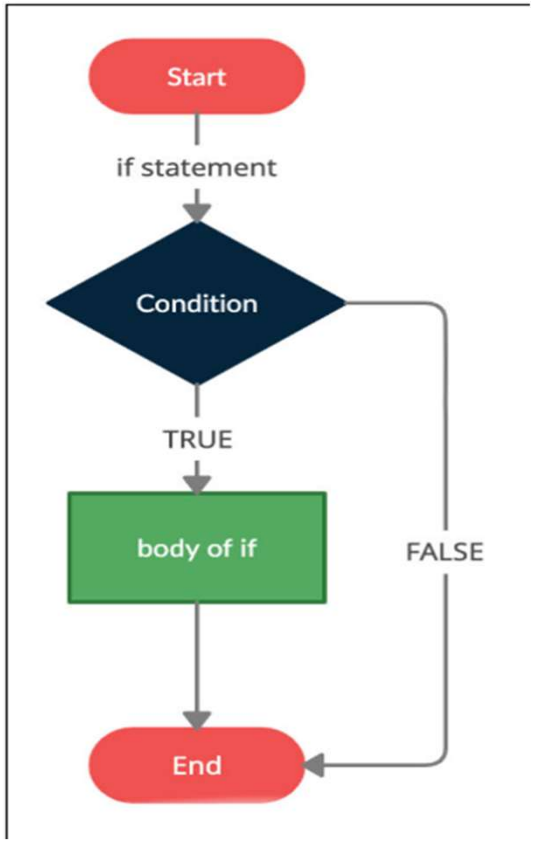
Python Flow Control

- **Python if Statement Syntax**

if test expression:

statement(s)

- The body of the if statement is indicated by the **indentation.**
- Body starts with an **indentation** and the first unindented line marks the end.



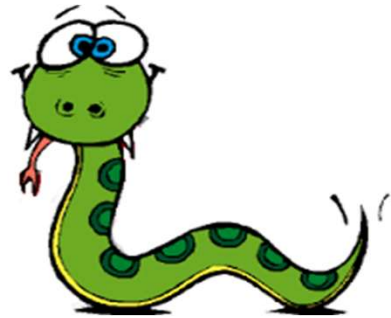
Python Flow Control

Python if Statement Syntax

```
>>> num = 3
>>> if num > 0:
        print(num, "is a positive number.")
3 is a positive number.
```

Example: Python if Statement

```
>>> num = 3
>>> if num > 0:
        print(num, "is a positive number.")
        print("testing")
3 is a positive number.
Testing
```

Examples

Program

```
# Python program to illustrate different if conditions
age = 16
limit = 25

if (age == 16):
    print('Age is correct as mentioned')

if (age < 25):
    print('Below age')

if (age > 10):
    print('Teen')

if (age <= limit):
    print('Under age limit')

if (age >= limit):  # will not be printed because the condition is FALSE
    print('Above limit')
```

Output

```
Age is correct as mentioned
Below age
Teen
Under age limit
```

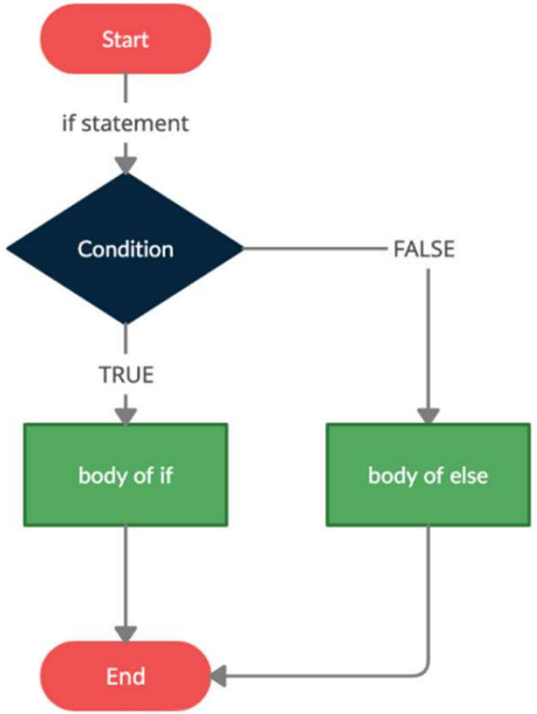
Python if...else Statement

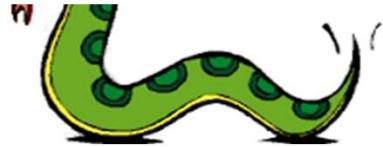
- Here, the program evaluates the **test expression** and will execute statement(s) only if the text expression is **True**. If the text expression is **False**, the statement(s) is not executed.
- In Python, the body of the **if** statement is indicated by the indentation. Body starts with an indentation and the first unindented line marks the end.
- Python interprets non-zero values as **True**. **None** and **0** are interpreted as **False**.

Python if...else Statement

Syntax of if...else:

```
if test expression:
        Body of if
else:
        Body of else
```

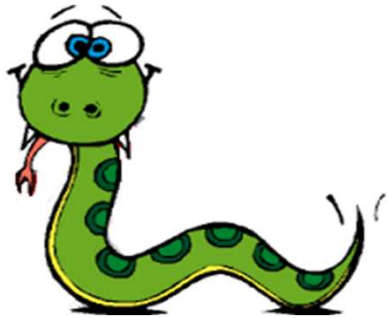




Program : if..else Statement

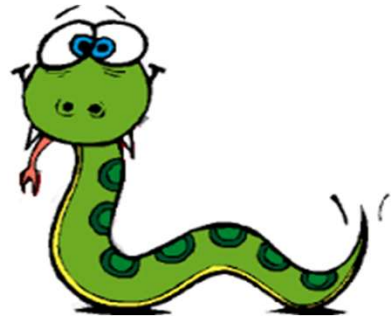
WAP to find input number is even or odd

Programming with Pyth



Solution : For if...else Statement Problem

```
num=int(input("Enter number"))
print("Checking number is even or odd")
if num %2 ==0:
        print("Number is Even")
else:
        print("Number is ODD")
```

Example

Program

```
# Python program to compare between subject marks
history = 45
science = 57

if (history > science):
    print('History marks greater than Science')
else:
    print('Science marks greater than History')
```

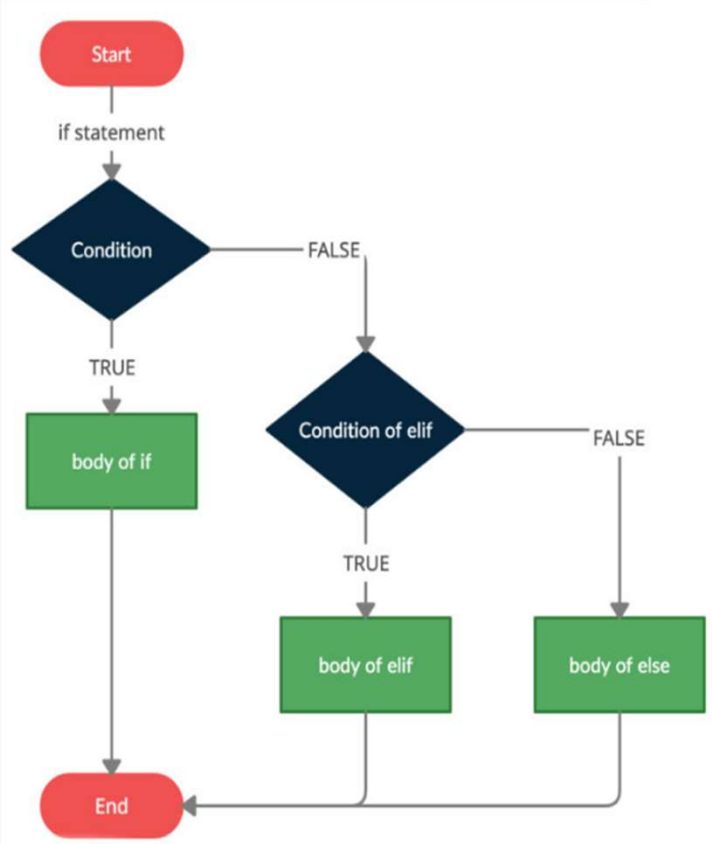
Output

```
Science marks greater than History
```

Python if...elif...else

Syntax of if...elif...else:

```
if test expression:
        Body of if
elif test expression:
        Body of elif
else:
        Body of else
```



Program :if...elif...else

- **WAP to display grade of student base on marks entered.**

70 > = distinction

60>= first class

50>=second class

otherwise pass class

Program Solution:if...elif...else

```
marks=int(input("Enter marks"))
if marks>= 70:
  print(" Distinction")
elif marks>=60:
  print(" First class")
elif marks>=50:
  print(" Second class")
else:
  print("Pass")
```

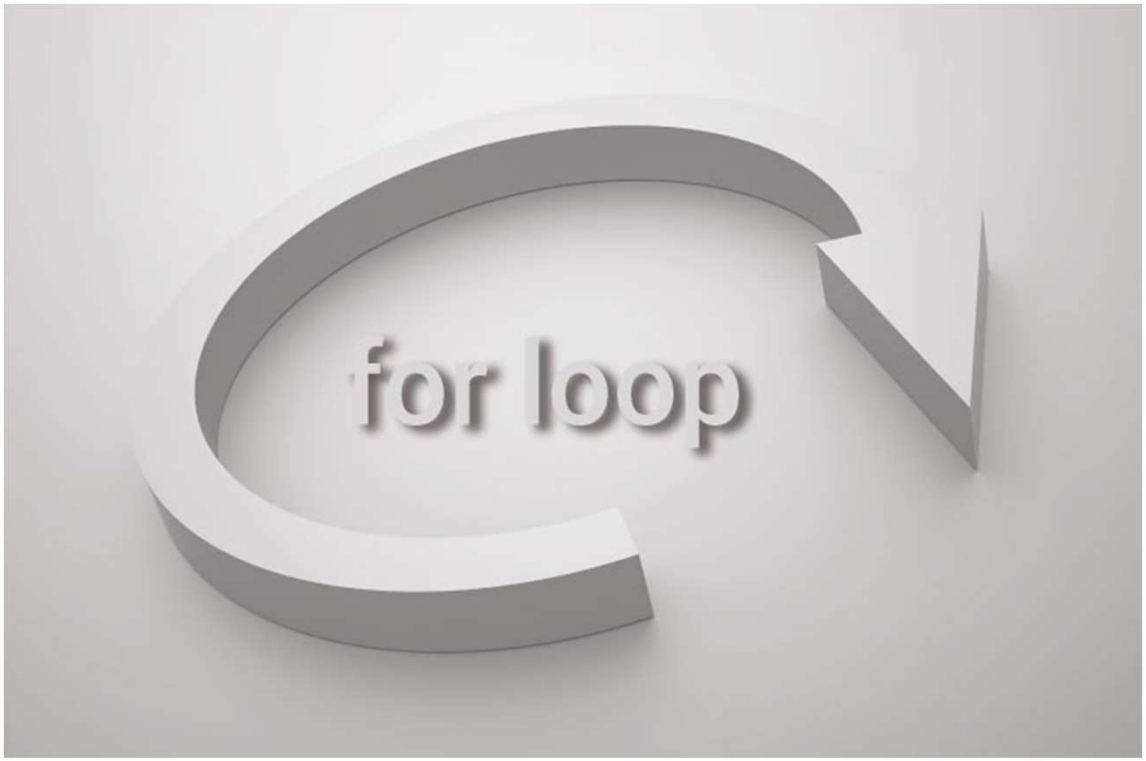
Python Nested if...else

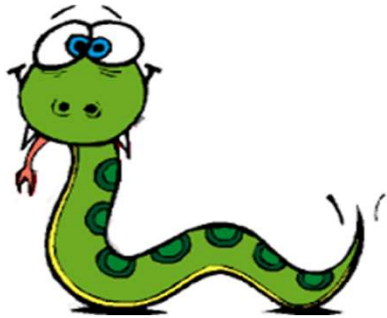
- We can have a if...elif...else statement inside another if...elif...else statement. This is called nesting in computer programming.
- Any number of these statements can be nested inside one another. Indentation is the only way to figure out the level of nesting.

WAP to find number is positive, negative or zero.

```
num = float(input("Enter a number: "))
if num >= 0:
        if num == 0:
                print("Zero")
        else:
                print("Positive number")
else:
        print("Negative number")
```

Loops – for loop





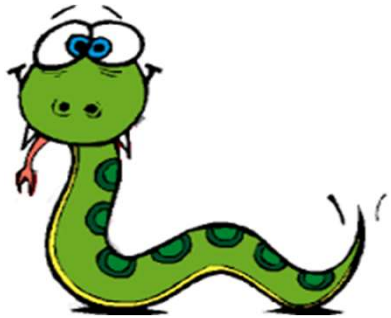
Loops – for loop

- The **for loop** in Python is used to iterate over a sequence (list, tuple, string) or other iterable objects.
- Iterating over a sequence is called traversal.
- **Syntax of for Loop**

for **Val** in **sequence**:

Body of for

Variable name **sequence name**



Loops – for loop

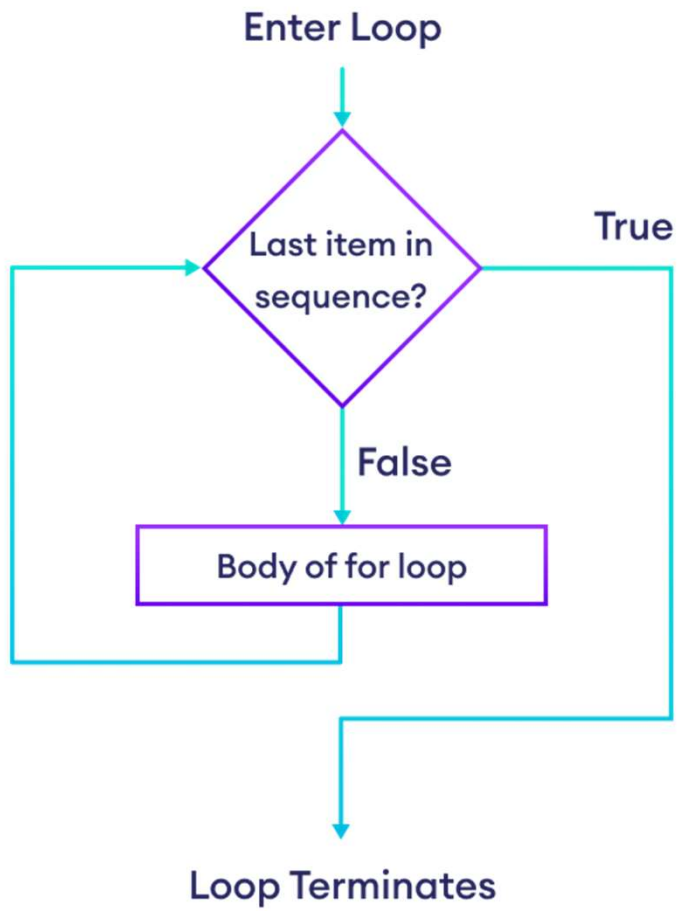
Syntax of for Loop

for Val in sequence:

    Body of for

- Here, **Val is variable** that takes the value of the item inside the sequence on each.
- Loop continues **until we reach the last item** in the sequence. The body of for loop is separated from the rest of the code using indentation

Flowchart for For Loop



Python for Loop

- In Python, a for loop is used to iterate over sequences such as lists, strings, tuples, etc.

Program

```
languages = ['Swift', 'Python', 'Go']

# access elements of the list one by one
for i in languages:
    print(i)
```

Output

```
Swift
Python
Go
```

Example: Python for Loop

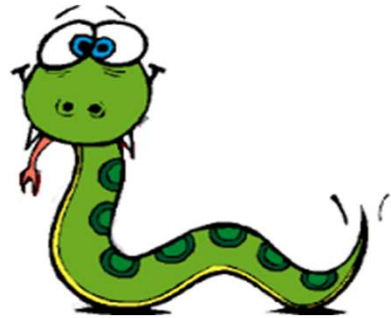
```
>>>numbers = [6,5,3,8,4,2,5,4,11]
# variable to store the sum
>>>sum = 0
# iterate over the list
>>>for val in numbers:
                    sum = sum+val
>>>print("The sum is",sum)
```

The sum is 48

The range() function

- We can generate a sequence of numbers using **range()** function.
- **range(n)** - will generate numbers from 0 to n-1 (n numbers).
- **range(start, stop, step size)** - We can also define the start, stop and step size as:

range(start, stop, step size)- step size defaults to 1 if not provided.



The range() function

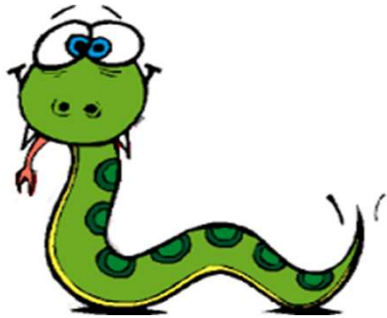
Note :

This function does not store all the values in memory, it would be inefficient. So it remembers the **start, stop, step size** and generates the next number on the go.

To force this function to output all the items, we can use the function **list().**

Example of Range


```
>>> print(range(0, 10))
range(0,10)
>>> print(list(range(10)))
[0,1,2,3,4,5,6,7,8,9]
>>> print(list(range(2,20,3)))
[2, 5, 8, 11, 14, 17]
```

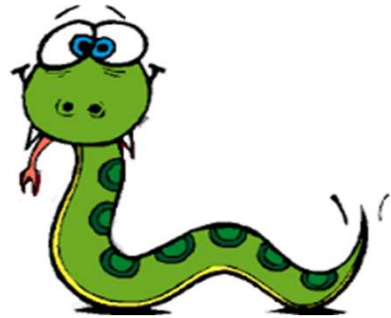
The range() function in for loop

- We can use the range() function in for loops to iterate through a sequence of numbers.
- It can be combined with the len() function to iterate though a sequence using indexing.

```
>>> genre = ['pop', 'rock', 'jazz']
>>> for i in range(len(genre)):
        print("I like", genre[i])
I like pop
I like rock
I like jazz
```

for loop with else

- A for loop can have an optional else block as well. The else part is executed if the items in the sequence used in for loop exhausts.
- break statement can be used to stop a for loop. In such case, the else part is ignored.
- Hence, a for loop's else part runs if no break occurs



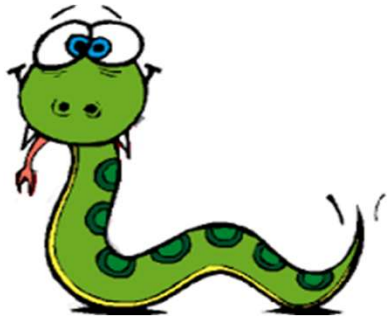
for loop with else


```
list_of_digits = [0,1,2,3,4,5,6]
input_digit = int(input("Enter a digit: "))
for i in list_of_digits:
    if input_digit == i:
        print("Digit is in the list")
        break
else:
        print("Digit not found in list")
```

Exercise on For Loop

```
# Addition of first ten numbers
sum=0
for i in range(1,10):
   sum+=i
print("sum is",sum)
```

Output:45



Exercise on For Loop

WAP to print below pattern

```
*
**
***
****
*****
```

Solution for Pattern

```
for i in range(0, 5):
    for j in range(0, i+1):
        print("* ",end="")
    print()
```

Exercise on For Loop

WAP to print below pattern

A

BB

CCC

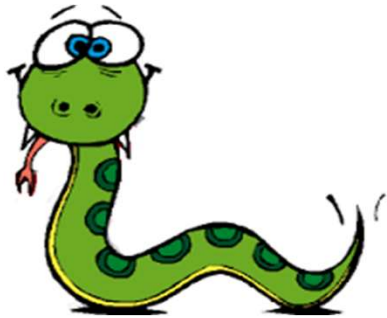
DDDD

EEEEE

Solution for Pattern

```
val = 65
for i in range(0, 5):
    for j in range(0, i+1):
        ch = chr(val)
        print(ch, end=" ")
    val = val + 1
    print()
```

Loops- While Loop



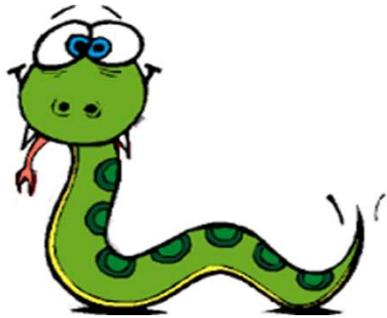
Loops- While Loop

- **The while loop in Python is used to iterate over a block of code as long as the test expression (condition) is true.**

- **We generally use this loop when we don't know beforehand, the number of times to iterate.**

- **Syntax of while Loop in Python**

```
while test_expression:
    Body of while
```

Flow Chart

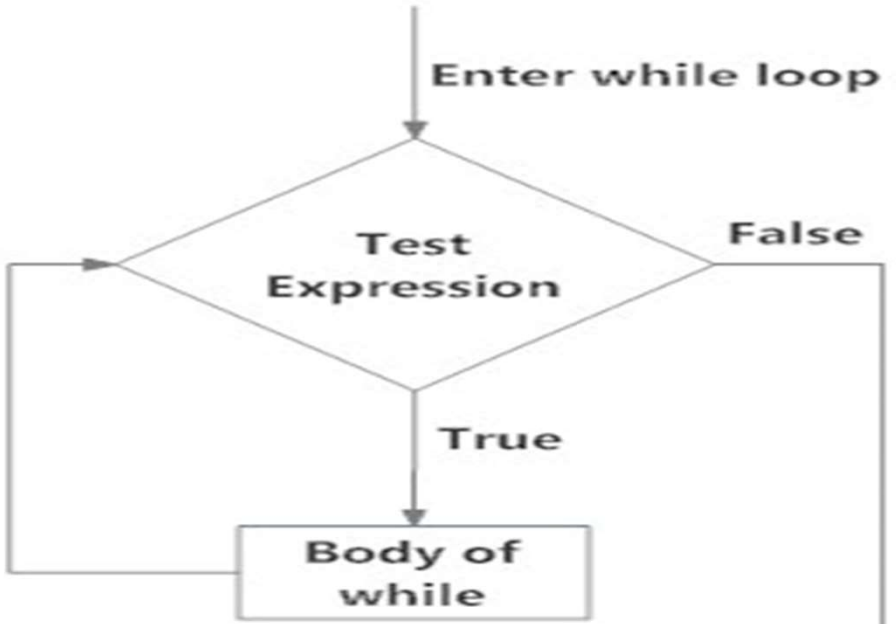
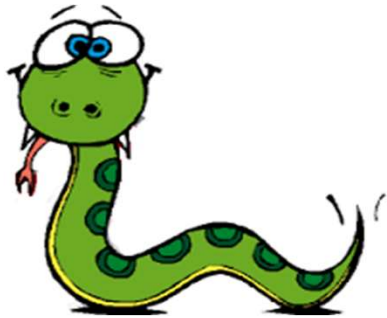


Fig: operation of while loop



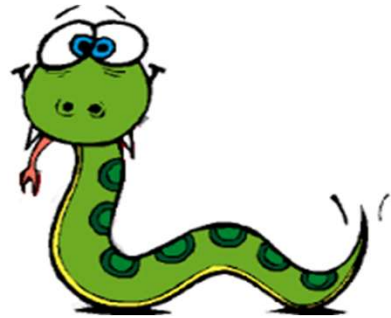
Example

```
# Program to add natural  numbers upto
# sum = 1+2+3+...+n
# To take input from the user,
#n = int(input("Enter n: "))
n = 10
# initialize sum and counter
sum = 0
i = 1
while i <= n:
        sum = sum + i
        i = i+1    # update counter
# print the sum
print("The sum is", sum)
```

while loop with else

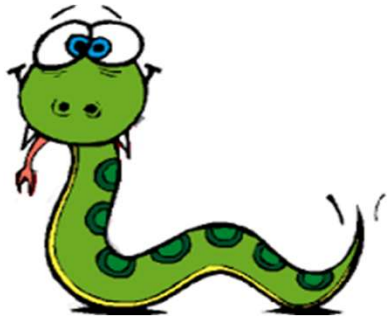
- Same as that of for loop, we can have an optional **else** block with while loop as well.
- The **else** part is executed if the condition in the while loop evaluates to **False**.
- The while loop can be terminated with a break statement.
- In such case, the **else** part is ignored.

Hence, a while loop's **else** part runs if no break occurs and the condition is false.



Example of while loop with else

```
counter = 0
while counter < 3:
        print("Inside loop")
        counter = counter + 1
else:
        print("Inside else")
```

Loop Control Statements

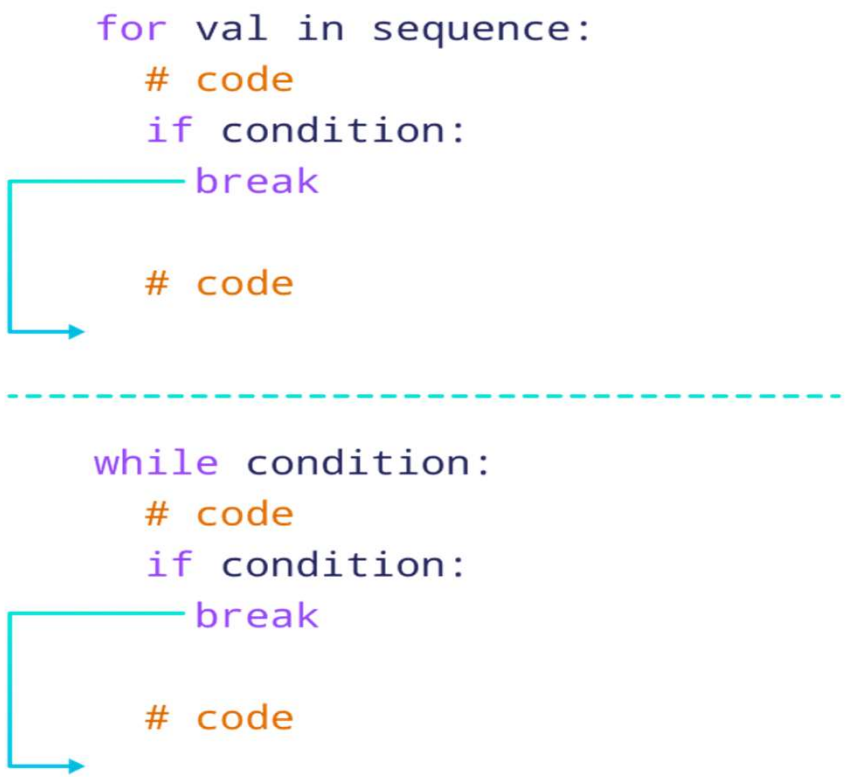
break	Jumps out of the closest enclosing loop
continue	Jumps to the top of the closest enclosing loop
pass	Does nothing, empty statement placeholder

break statement

- The break statement terminates the loop containing it.
- Control of the program flows to the statement immediately after the body of the loop.
- If break statement is inside a nested loop (loop inside another loop), break will terminate the innermost loop.

e.g.
```
for val in "string":
        if val == "i":
                break
        print(val)
print("The end")
```

Working of Python break Statement



Example

```
for i in range(5):
    if i == 3:
        break
    print(i)
```

```
0
1
2
```

Continue

- The continue statement is used to skip the rest of the code inside a loop for the current iteration only.
- Loop does not terminate but continues on with the next iteration.

Syntax of Continue

```
continue
```

e.g.

```
for val in "string":
        if val == "i":
                continue
        print(val)
print("The end")
```

Working of continue Statement in Python

```
for val in sequence:
    # code
    if condition:
        continue

    # code
```

```
while condition:
    # code
    if condition:
        continue

    # code
```

Example

```
for i in range(5):
    if i == 3:
        continue
    print(i)
```

```
0
1
2
4
```

pass

- In Python programming, pass is a null statement. The difference between a comment and pass statement in Python is that, while the interpreter ignores a comment entirely, pass is not ignored.
- However, nothing happens when pass is executed. It results into no operation (NOP).
- **Syntax of pass**

  pass
- We generally use it as a placeholder.

pass

- Suppose we have a loop or a function that is not implemented yet, but we want to implement it in the future. They cannot have an empty body. The interpreter would complain. So, we use the pass statement to construct a body that does nothing.

e.g. sequence = {'p', 'a', 's', 's'}

```
for val in sequence:
        pass
```

Example

```
if 1 + 2 == 3:
  print("Correct math")
  pass
  print("This will also be printed.")
```

```
Correct math
This will also be printed.
```

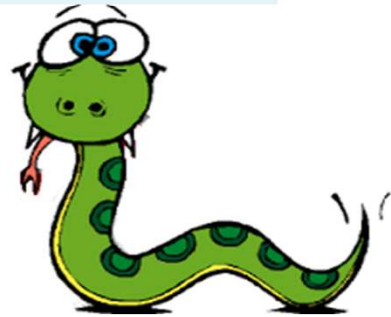
Exercise on Loops

- An Armstrong number of **three** digits is an integer such that the sum of the cubes of its digits is equal to the number itself.
- For example, **371** is an Armstrong number since 3**3 + **7****3 + 1**3 = **371**.
- Write a program to find all Armstrong number in the range of 0 and **999**.


```
# Program to check Armstrong numbers in a certain interval
lower = 100
upper = 2000
for num in range(lower, upper + 1):
   # order of number
   order = len(str(num))
   # initialize sum
   sum = 0
   temp = num
   while temp > 0:
       digit = temp % 10
       sum += digit ** order
       temp //= 10
   if num == sum:
       print(num)
```

Python Functions

Python Functions



FUNCTIONS

- A **function** is a **block of organized, reusable code** that is used to perform a **single,** related **action**.
- **Functions** provide **better modularity** for the applications.
- **Functions** provide a **high degree of code reusing**.

TYPES OF FUNCTION

THERE ARE TWO TYPES OF FUNCTION IN PYTHON

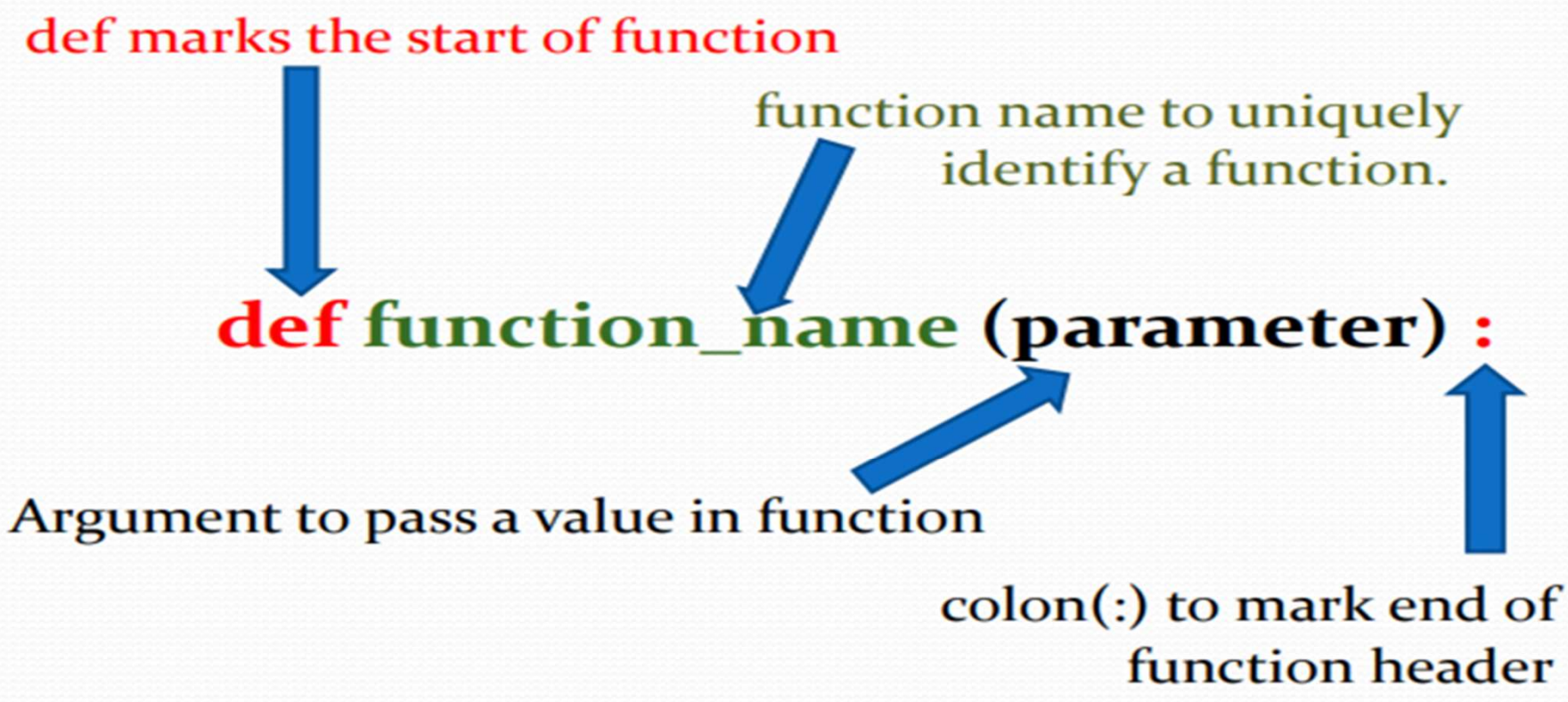
- BUILT-IN FUNCTION

Example: print(), input(), eval().

- USERDEFINE FUNCTION

Example:
```
def my_addition(x,y):
    sum = x + y
    return sum
```

DEFINING FUNCTION



RULES FOR DEFINING FUNCTION IN PYTHON

- Function blocks begin with the **keyword def** followed by the **function name and parentheses (( )).**
- Any **input parameters or arguments** should be placed **within these parentheses**. We also define **parameters inside** these **parentheses.**
- The **code block** within every function **starts** with a **colon (:)** and is **indented.**
- The **statement return** [expression] **exits a function**, optionally **passing back an expression** to the **caller.**
- A **return statement** with **no arguments** is the **same** as **return None**.

Function Syntax

Input Parameter is placed within the **parenthesis()** and also **define parameter inside** the **parenthesis.**

The keyword **def** introduces a **function definition.**

```
def function_name( parameters ):
   statement 1...
   Statement 2...
   .....................
   return [expression]
```

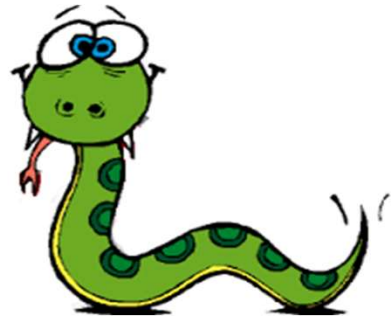
The **code block** within every **function starts** with a **colon(:)** .

Return statement exits a **function** block. And we can also **use return** with **no argument.**

Python Functions

Syntax of Function

```
def function_name(parameters):
        """docstring"""
        statement(s)
```

EXAMPLE OF FUNCTION

```
def greet(name):
        print("Hello,"+name+".Good morning!")
```

CALLING A FUNCTION

HOW TO CALL A FUNCTION?

Once we have defined a function, we can call it from another function, program or even the Python prompt.

To call a function we simply type the function name with appropriate parameters.

```
>>> greet(“everyone")

>>>Hello, everyone. Good afternoon!
```

EXAMPLE

```
def my_func():
    x = 10
    global y
    y = 20
    print("Value of y inside function:",x)
    print("Value of y inside function:",y)

x = 20
y = 10
my_func()
print("Value of y outside function:",x)
print("Value of y outside function:",y)
```

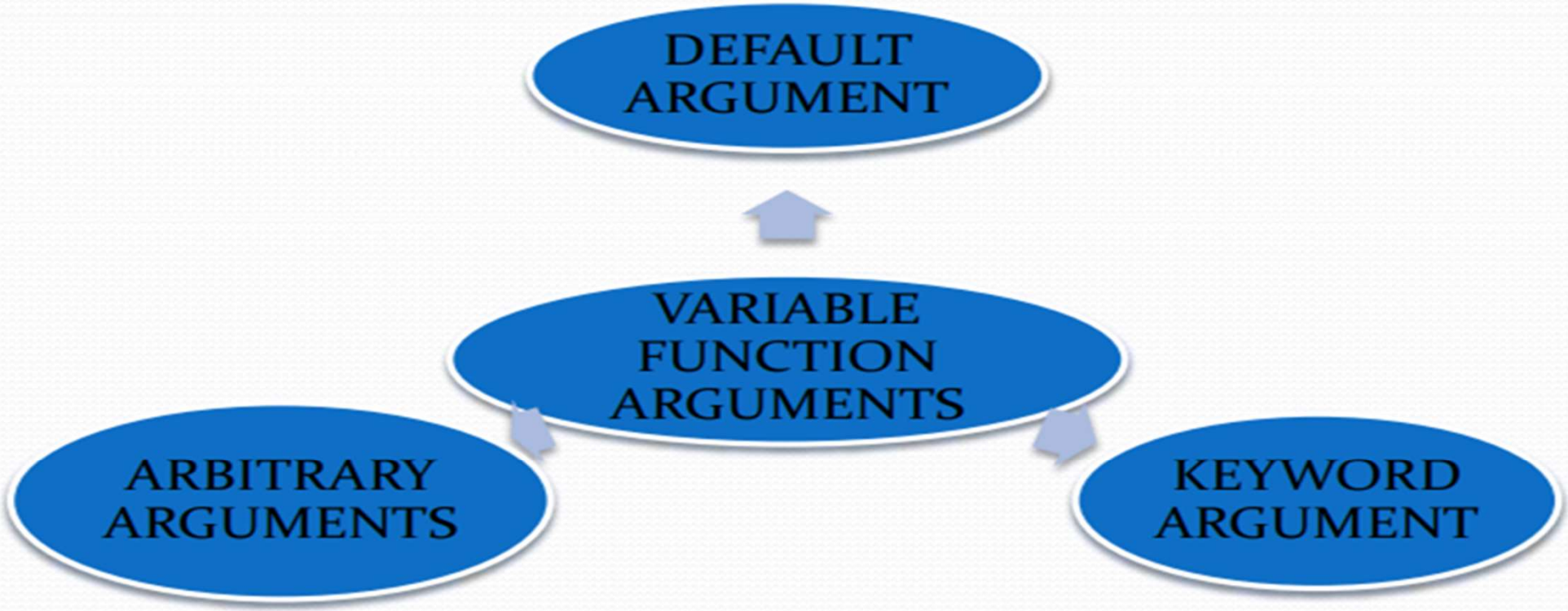
Example

```
def fun1():
    x=10
    global y
    y=20
    print("x inside fun:=",x)
    print("y inside fun:",y)
x=20
y=10
fun1()
print("x outside fun:=",x)
print("y outside fun:",y)
```

- **Output:**

```
x inside fun:= 10
y inside fun: 20
x outside fun:= 20
y outside fun: 20
```

FUNCTION ARGUMENTS



Positional Argument

- **Required arguments** are the arguments passed to a function in correct **positional order**.
- Here, the number of arguments in the function call should **match** exactly with the function definition.

```
def message(a,b):
        print(a,b)
```

Function call: message(12,34)

12 34

DEFAULT ARGUMENTS

- Default value to function argument is passed using assignment operator ' = '.

Example:

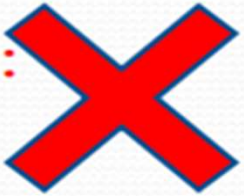
```
def greet(name, msg = "Good morning!"):
print("Hello, " name + ', ' + msg)
```

- Non-default argument cannot follow default argument

Example: def greet(msg = "Good morning!", name):

SyntaxError:

non-default argument follows default argument




```
>>> greet("John")
Hello,John,Good morning
>>> greet("John","Good Afternoon")
Hello,John,Good Afternoon
```

KEYWORD ARGUMENTS

- When we call a function with some values, these values get assigned to the arguments according to their position .
- Keyword arguments follows positional argument.

EXAMPLE OF KEYWORD ARGUMENT

- greet(name = "Bruce",msg = "How do you do?")

  → **2 keyword arguments**

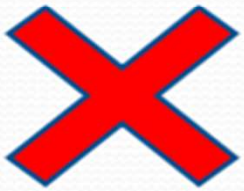
- greet(msg = "How do you do?",name = "Bruce")

  → **2 keyword arguments (out of order)**

- greet("Bruce",msg = "How do you do?")

  → **1 positional, 1 keyword argument**

- greet(name="Bruce","How do you do?")



SyntaxError:

non-keyword arg after keyword arg

ARBITRARY ARGUMENTS

- If number of arguments unknown we use arbitrary arguments
- (*) Asterisk before arguments define arbitrary arguments.

Example:

```
def greet(*names):
        for name in names:
         print("Hello",name)
>>>greet("Monica","Luke","Steve","John")
```

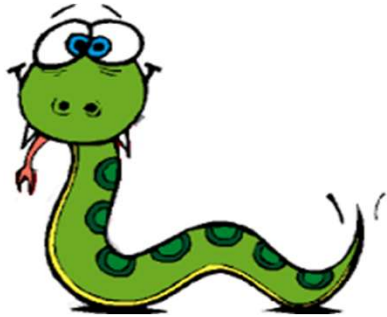

```
>>> greet("Monica","Luke","Steve","John")
Hello Monica
Hello Luke
Hello Steve
Hello John
```

RECURSION FUNCTION

- Like other programming language in python, function can call itself.
- Definition of recursion in python is same as other programming lang. -> C, C++, C#, Java etc.

Python Functions :Recursion

- Recursion is the process of defining something in terms of itself.
- **Advantages of recursion**
  - Recursive functions make the code look clean and elegant.
  - A complex task can be broken down into simpler sub-problems using recursion.
  - Sequence generation is easier with recursion than using some nested iteration.
- **Disadvantages of recursion**
  - Sometimes the logic behind recursion is hard to follow through.
  - Recursive calls are expensive (inefficient) as they take up a lot of memory and time.
  - Recursive functions are hard to debug



EXAMPLE OF RECURSION

```
def recur_fact(x):
    if x == 1:
                return 1
else:
                return (x * recur_fact(x-1))

 num = int(input("Enter a number: "))
 if num >= 1:
print("The factorial of", num, "is", recur_fact(num))
```

```
def recur_fact(n):
    if n==1:
        return 1;
    else:
        return(n*recur_fact(n-1))
#print(fact)
n=int(input("enter"))
fact=recur_fact(n)
print(fact)
```

```
>>> enter 5
120
```

ANONYMOUS FUNCTION

- Function without function name.
- Function is defined using lambda keyword, hence function is also called Lambda function.
- Used for short period in python.
- Can have any number of arguments but only single expression.
- Specially used with built-in functions like filter(), map() etc.

EXAMPLE OF LAMBDA FUNCTION

- double = lambda x: x * 2 print(double(5))

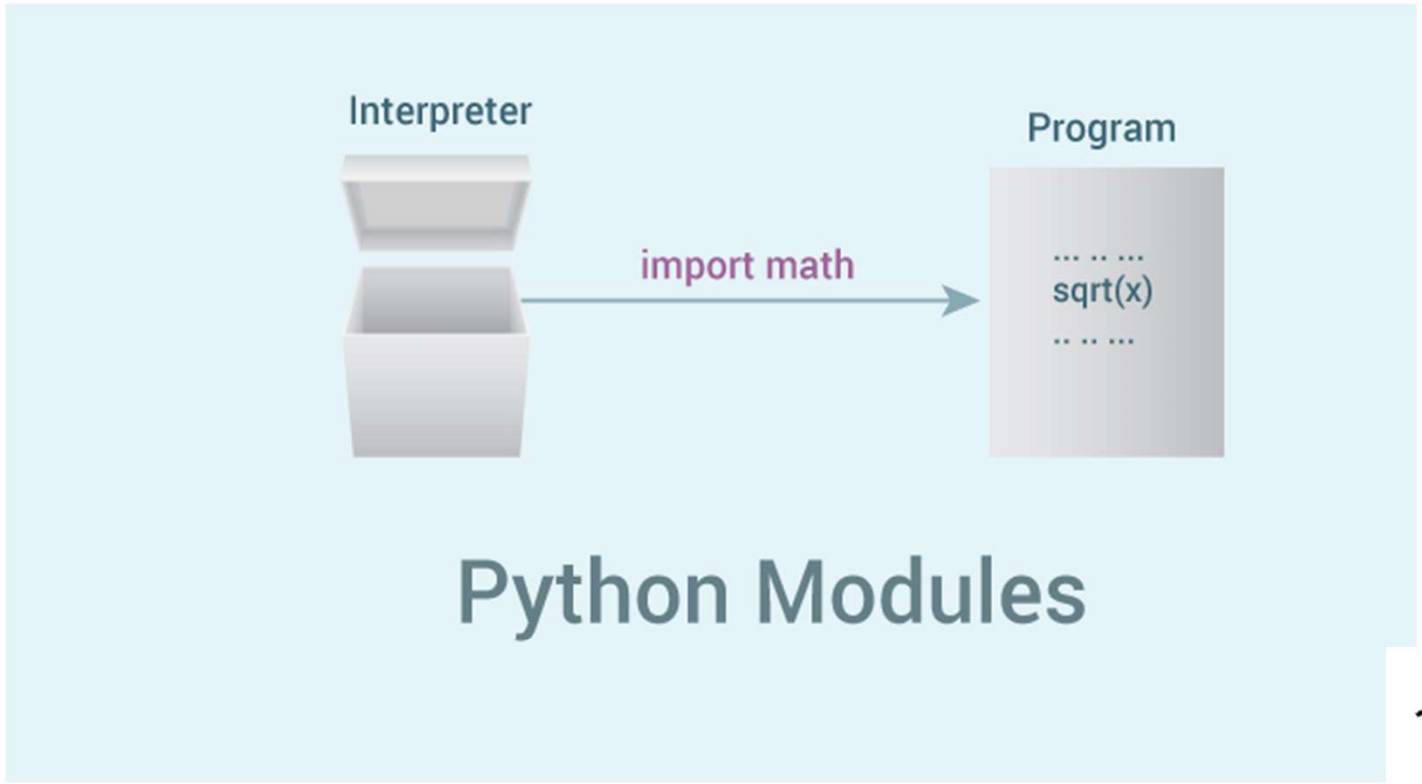
- my_list = [1, 5, 4, 6, 8, 11, 3, 12]

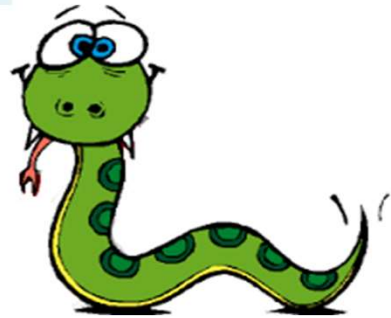
```
new_list = list(filter(lambda x: (x%2 == 0) , my_list))
        print(new_list)
```

```
res=lambda x:x*x
print(res(4))
Output: 16

new_list=[1,2,3,4,5]
even_list=list(filter(lambda x:x%2==0,new_list))
print(even_list)
[2,4]
```

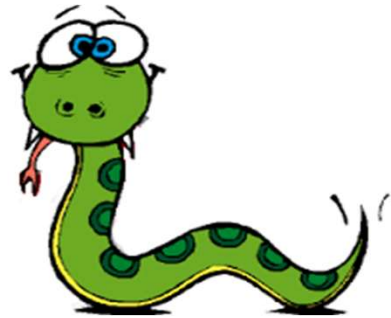
Python Modules





Python Module

- Modules in Python are simply Python files with the .py extension, which implement a set of functions. Modules are imported from other modules using the **import** command.
- A file containing Python code, for e.g.: example.py, is called a module and its module name would be example.
- To import a module, we use the **import** command.
- **import** modulename
- https://docs.python.org/2/library/



Python import statement

- We can import a module using **import statement** and access the definitions inside it using the dot operator.

e.g.

```
import math
print("The value of pi is", math.pi)
```

Output: The value of pi is 3.141592653589793

Import with renaming

- We can import a module by renaming it as follows.
- We have to **rename** the math module as **m**.

```
import math as m
print("The value of pi is", m.pi)
```

Output:

```
The value of pi is 3.141592653589793
```

Python from...import statement

- We can import specific names from a module without importing the module as a whole.

```
from math import pi
print("The value of pi is", pi)
```

Output:

The value of pi is 3.141592653589793

Import multiple values

- We could have imported multiple attributes as follows.

```
>>> from math import pi, e
>>> pi
3.141592653589793
>>> e
2.718281828459045
```

Import all names

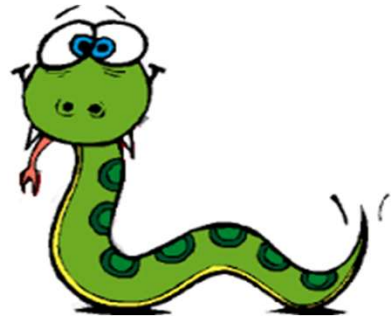
- We can import all names(definitions) from a module using the following construct.

```
>>>from math import *
>>>print("The value of pi is", pi)
3.141592653589793
>>>print("The value of pi is", e)
2.718281828459045
```

Python Module : Creating own Module

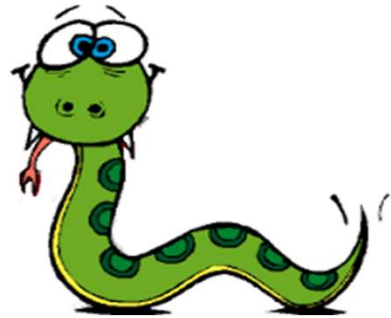
- **Save this code as test.py**

```
#define display function
def display():
    print(" Last session don't sleep")
#define fact function
def fact(n):
    prod=1
    for i in range(1,n+1,1):
        prod*=i
    return prod
```

Python Module :Using own module

```
#calculate factorial and display message using test module. Save it as another
    python file like fun1.py.
#Using the module name we can access the function using dot (.) operation
import fun_def
fun_def.display()
n=int(input("enter number"))
x=fun_def.fact(n)
print("Factorial is:",x)
```

Python Module :Using own module

#Run the fun1.py file

- **Output:**

Last session don't sleep

enter number5

Factorial is: 120

Python Module :Using own module

#Another type of import:-

```
from fun_def import *
display()
n=int(input("enter number"))
x=fact(n)
print("Factorial is:",x)
```

Python Module :Using own module

#Run the .py file

- **Output:**

Last session don't sleep

enter number5

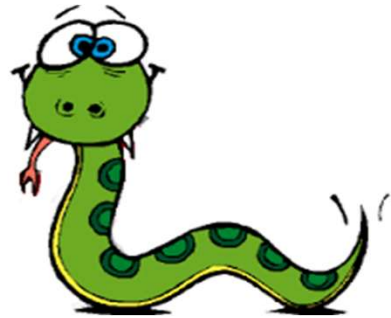
Factorial is: 120

Exercise Python Module

Develop a module with two functions

1. Find given number is prime number or not
2. Generate Fibonacci series

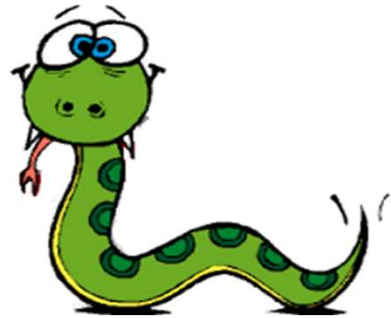
WAP to import these function in menu driven fashion.



Date & time

```
>>> import time
>>> localtime=time.localtime(time.time())
>>> print localtime

>>> localtime =
time.asctime(time.localtime(time.time()))
>>> print localtime
```



Calendar

```
>>> import calendar
>>> c=calendar.month(2018,1)
>>> print c
```

Help

```
>>> help(print)
>>> help(list)
>>> import math
>>> help(math.sin)
>>> help(math.cos)
```