



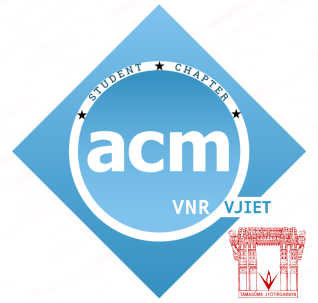
Python Programming

DAY 1

Introduction To Python

**Getting Started – Installation And Setup
Variables, Operators and Basic Datatypes**

Introduction - What Is Python?



- Python is an interpreted, high-level, general-purpose programming language.
- Python's design philosophy emphasizes code readability with its notable use of significant whitespace.
- Python is dynamically typed and garbage-collected.
- It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented, and functional programming.

Introduction – A Brief History

- Python was conceived in the late 1980s by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC language, capable of exception handling and interfacing with the Amoeba operating system. Its implementation began in December 1989.
- Python 2.0 was released on 16 October 2000 with many major new features, including a cycle-detecting garbage collector and support for Unicode.
- Python 3.0 was released on 3 December 2008. It was a major revision of the language.

Introduction – Why Python?

Comparison with other languages

C Program	Java Program	Python Program
<pre>main() { printf("hello, world\n"); }</pre>	<pre>class myfirstjavaprogram { public static void main(String args[]) { System.out.println("Hello World!"); } }</pre>	<pre>print ("Hello World!!")</pre>

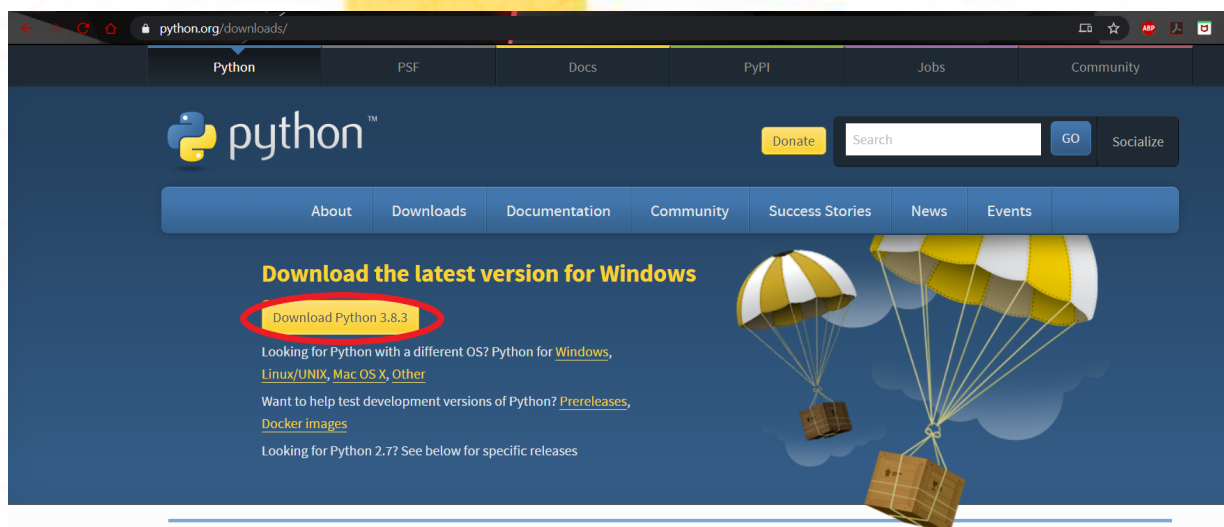


Introduction – Why Python?



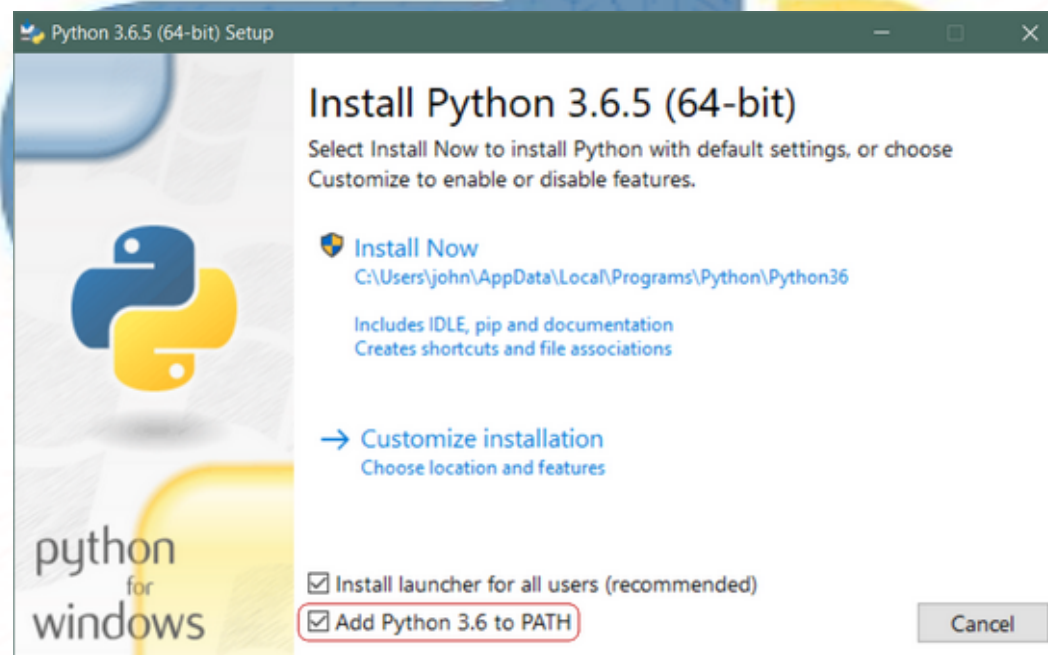
Getting Started - Installation

- Python is a free software, and the latest build (3.9.3) can be downloaded from <https://www.python.org/downloads/>. (The link will be provided in the chat).
- Python is cross platform compatible, i.e. it is supported by both Windows and Mac OS.



Getting Started - Installation

- Once the file has been downloaded to your local storage, run the setup. It should open up a screen similar to the one depicted.

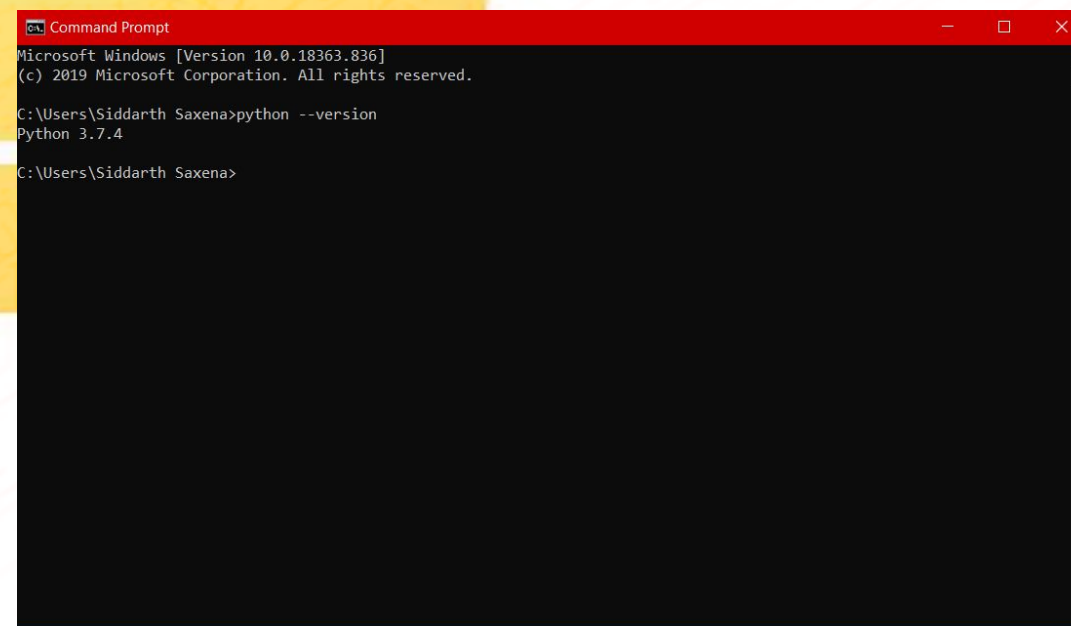
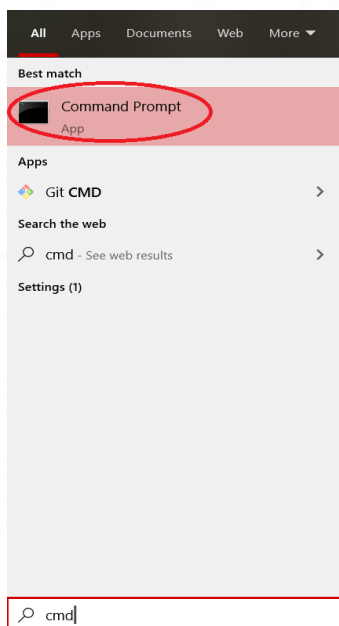


NOTE: Version Number may differ.

IMPORTANT SETUP POINT: Make sure that the “ADD Python to PATH” Field is checked. If missed, watch this video : https://www.youtube.com/watch?v=4V14G5_CNGg

Getting Started - Installation

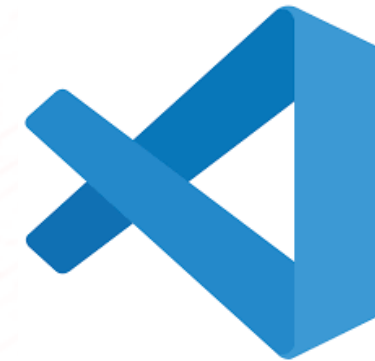
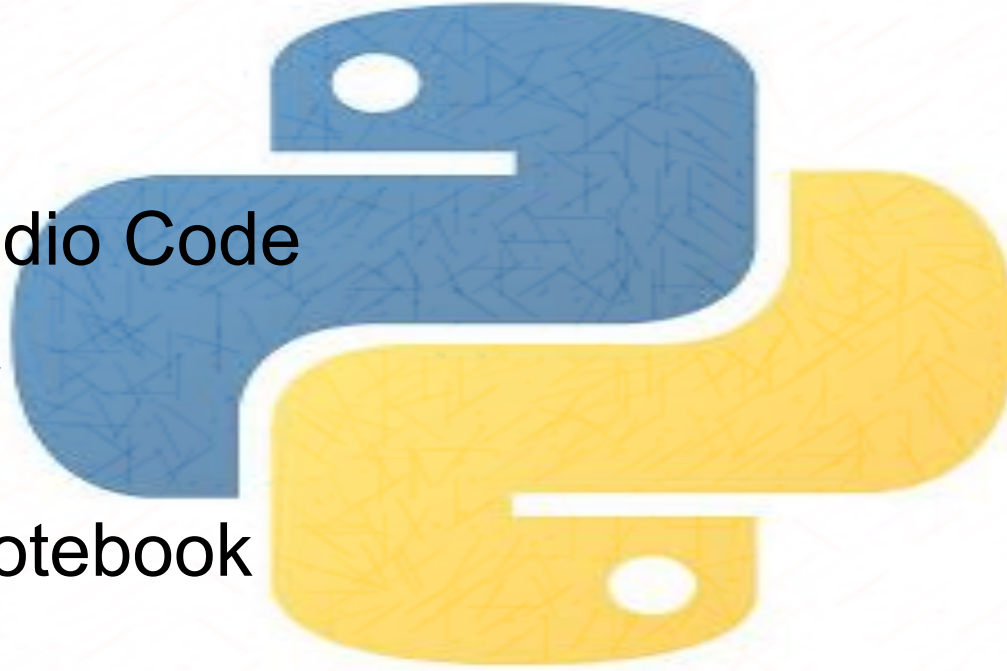
- Click “Install Now” and wait for the setup to configure Python on your system.
- After this, you can check the validity of your install by opening Windows Command Prompt and typing in “python --version”.



Different IDEs for Python



- Visual Studio Code
- Anaconda
- Pycharm
- Jupyter Notebook
- IDLE



Variables

- A **Variable** is something which can be changed. It contains values stored in it. These values are referred to as **Literals**.

Syntax:

<varname> = value

Here ' = ' acts as the **assignment** operator. It stores the value in variable.

Example:

1). a = 5

2). b="hello"

Basic Datatypes

- **Basic Datatypes:** There are five basic datatypes in python. They are:
- **Integer**
- **Float**
- **Complex**
- **String**
- **Boolean**



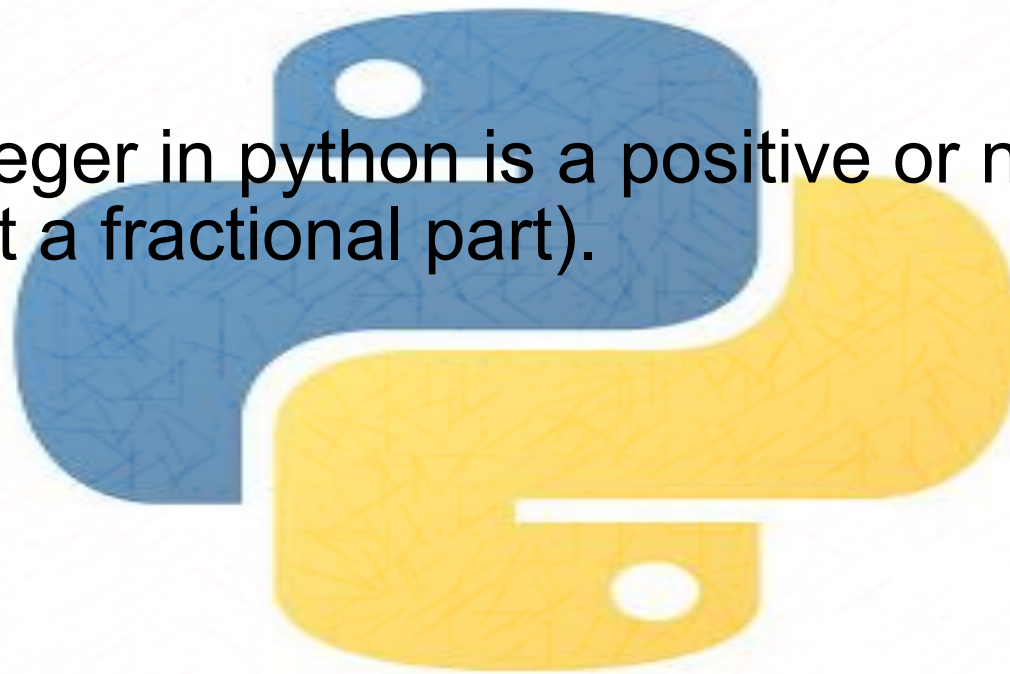
Basic Datatypes - Integer

- **Integer:** An integer in python is a positive or negative whole number(without a fractional part).

Example:

1). $A = 5$

2). $N = 15$



Basic Datatypes - Float

- **Float:** Any real number with a floating point representation in which a fractional component is denoted by a decimal symbol or scientific notation, can be termed to be of the type “Float” in python.

Example:

1). $D = 3.14$

2). $F = 15.6789$

Basic Datatypes - Complex

- **Complex:** The complex datatype in python is used to represent any number with a real and imaginary component as **$x+yj$** . x and y are floats and j^2 is -1 (square root of -1 called an imaginary number)

Example:

1). $C = 3+4j$

2). $Z = 9.26+8.14j$

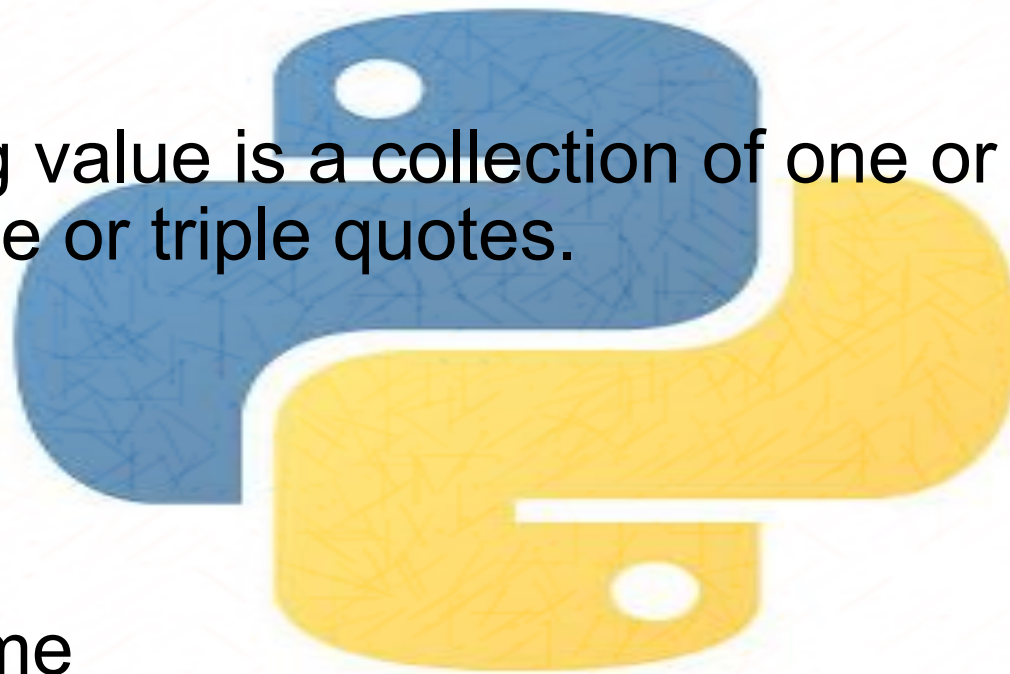
Basic Datatypes - String

- **String:** A string value is a collection of one or more characters put in single, double or triple quotes.

Example:

1). B = 'python'

2). A = """Welcome
to
programming."""



Basic Datatypes - Boolean

- **Boolean:** Data with one of two built-in values, **True** or **False**. Notice that 'T' and 'F' are capital. **true** and **false** are not valid booleans and Python will throw an error for them.

Example:

- 1). `check = True`
- 2). `Flag = False`

Operators

- There are seven types of operators in Python. They are:
- **Arithmetic operators:** (+, -, *, /, //, %, **)
- **Assignment operators:** (=, +=, -=, *=, /=, //=, **=, \=, ^=, >>=, <<=)
- **Comparison operators:** (==, !=, >, >=, <, <=)
- **Logical operators:** (and, or, not)
- **Membership operators:** (in, not in)
- **Identity operators:** (is, is not)
- **Bitwise Operators:** (&, |, ^, ~, <<, >>)



Python Programming

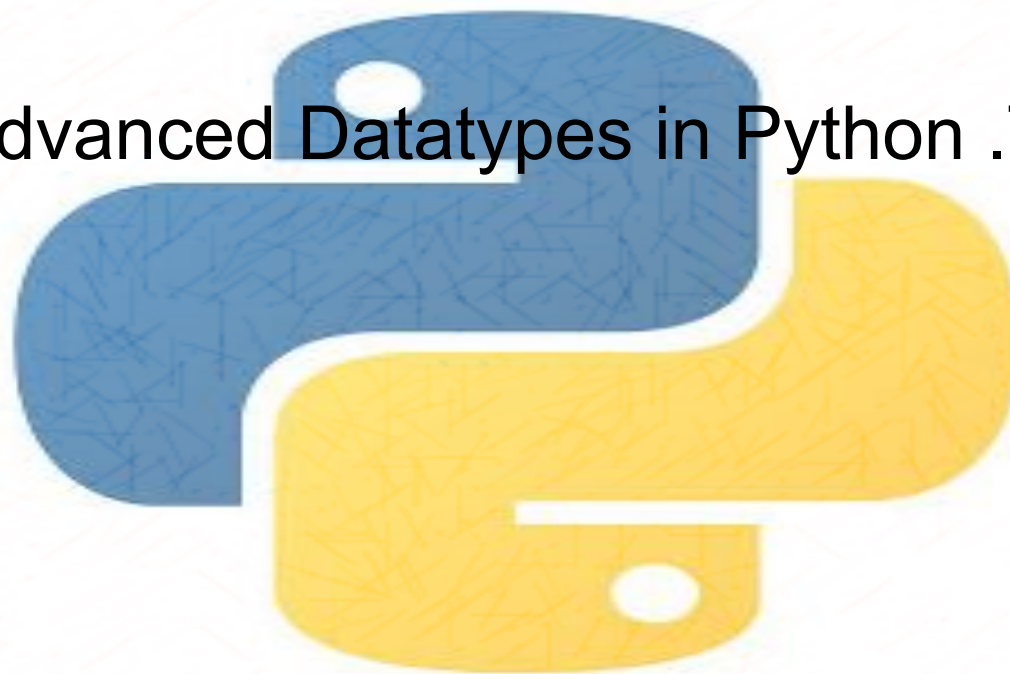
DAY 2

Advanced Datatypes

Advanced Datatypes

There are four Advanced Datatypes in Python .They are:

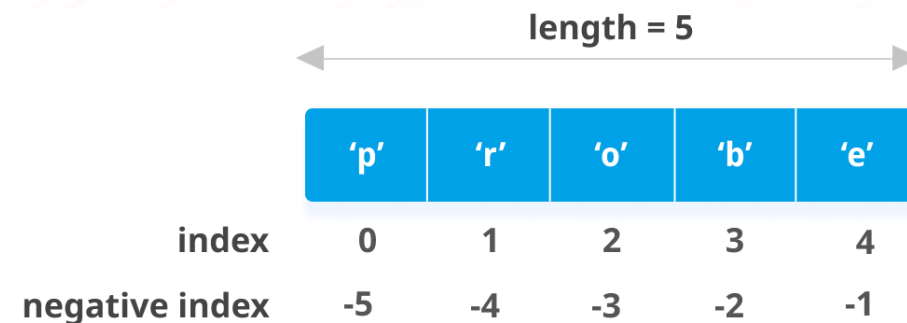
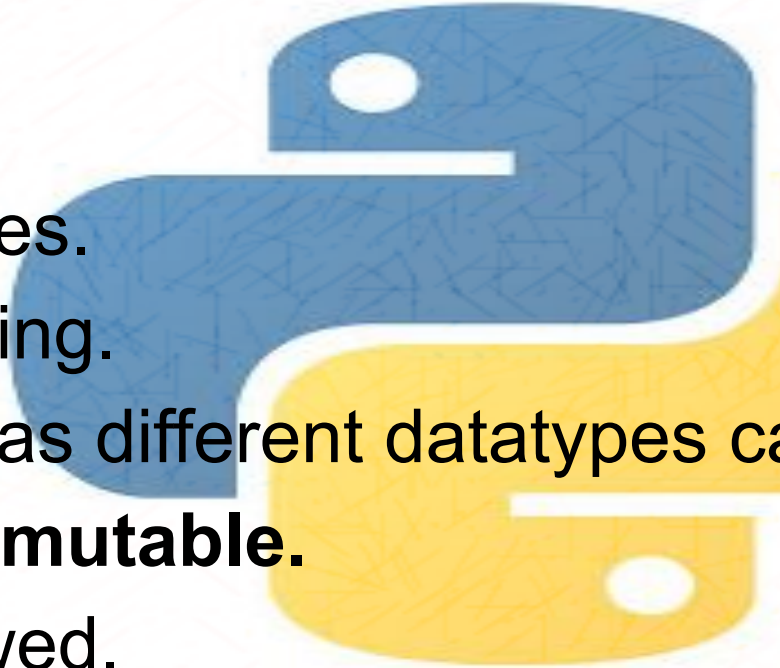
- List
- Tuple
- Set
- Dictionary



Advanced Datatypes - List

List:

- Allows duplicates.
- Supports indexing.
- Similar as well as different datatypes can be stored.
- **List items are mutable.**
- Nesting is allowed.
- Enclosed in square brackets, separated by the character ,



	length = 5				
	'p'	'r'	'o'	'b'	'e'
index	0	1	2	3	4
negative index	-5	-4	-3	-2	-1

List Methods

The following is a list of all the methods for a **List** object:

- **list.append(x)**: Adds an item to the end of the list.
- **list.extend(iterable)**: Extends the list by appending all the items from the iterable.
- **list.insert(i, x)**: Inserts an item at a given position. The first argument is the index of the element before which to insert, i.e. `a.insert(0, x)` inserts at the front of the list.

List Methods

- **list.pop([i]):** Removes the item at the given position in the list, and returns it. If no index is specified, a.pop() removes and returns the last item in the list.
- **list.remove(x):** Removes the first item from the list whose value is equal to x. It raises a ValueError if there is no such item.
- **list.clear():** Removes all items from the list.
- **Del list[x]:** Removes the item x from the list.

List Methods

- **len():** Prints the length of the list.
- **max():** Finds the maximum element in the list and prints it.
- **min():** Searches for the minimum element in the list and prints it.
- **sum():** Returns the sum of all list values.
- **list.index(x):** Returns zero-based index in the list of the first item whose value is equal to x. Raises a ValueError if there is no such item.

List Methods

- **list.count(x):** Returns the number of times x appears in the list.
- **list.sort(key=None, reverse=False):** Sorts the items of the list in place (the arguments can be used for sort customization).
- **list.reverse():** Reverses the elements of the list in place.
- **list.copy():** Returns a shallow copy of the list.

Advanced Datatypes - Tuple

Tuple:

- Allows duplicates.
- Supports indexing.
- Similar as well as different datatypes can be stored.
- **Tuple items are immutable.**
- Nesting is allowed.
- Read only version of list.
- Enclosed in parenthesis ,separated by the character ,

Tuple = (0, 1, 2, 3, 4, 5)

0	1	2	3	4	5
---	---	---	---	---	---

Tuple[0] = 0 Tuple[0:] = (0, 1, 2, 3, 4, 5)

Tuple[1] = 1 Tuple[:] = (0, 1, 2, 3, 4, 5)

Tuple[2] = 2 Tuple[2:4] = (2, 3)

Tuple[3] = 3 Tuple[1:3] = (1, 2)

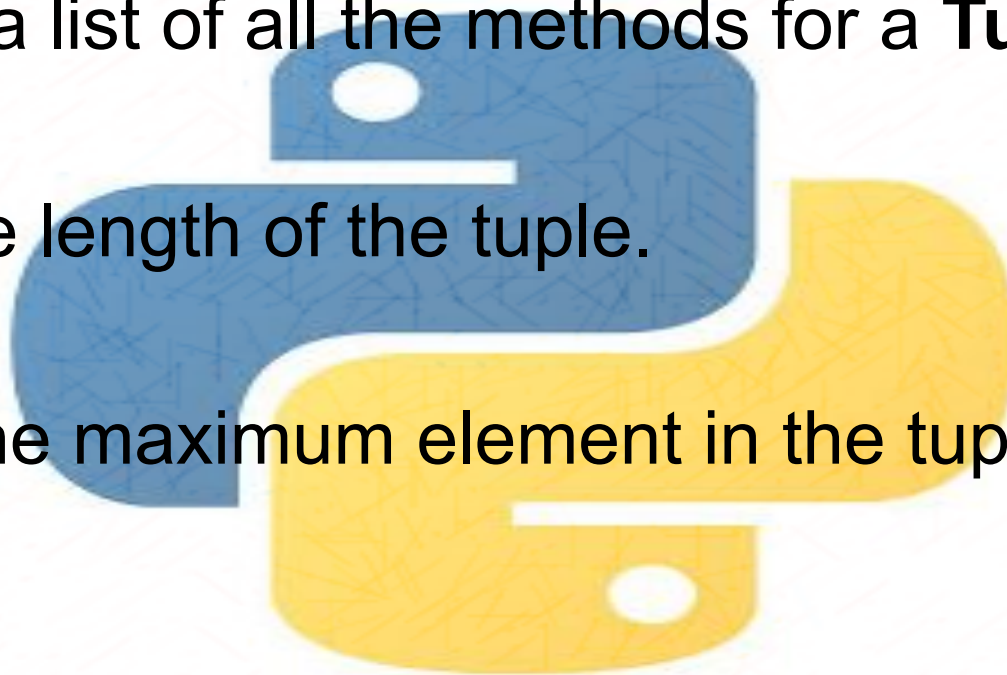
Tuple[4] = 4 Tuple[:4] = (0, 1, 2, 3)

Tuple[5] = 5

Tuple Methods

The following is a list of all the methods for a **Tuple** object:

- **len()**: Prints the length of the tuple.
- **max()**: Finds the maximum element in the tuple and prints it as the output.
- **min()**: Searches for the minimum element in the tuple and prints it as the output.



Tuple Methods

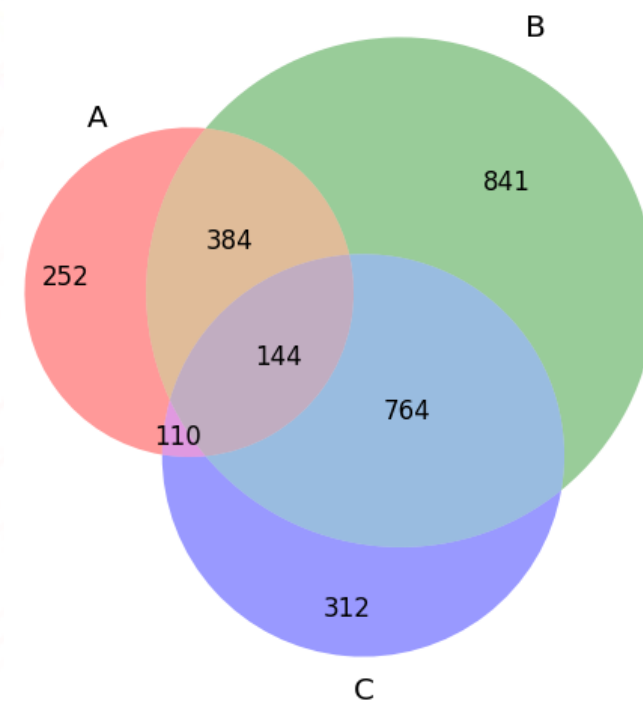
- **tuple.count(x)**: Returns the number of times the value of **x** occurs in the tuple.
- **tuple.index(x)**: Searches the tuple for the specified value of **x** and returns the position of where it was found.

As Tuple objects are **immutable**, methods such as **append()**, **extend()**, **insert()**, **remove()**, **pop()**, **reverse()**, **copy()**, **sort()**, **sorted()**, **clear()** are not defined. Furthermore, tuples **do not support single item-wise deletion**. However, a tuple can be entirely deleted using the **del()** function.

Advanced Datatypes – Set

Set:

- If we wish to represent a group of element values as a single entity then we opt for a set.
- Duplicates are not allowed.
- Insertion order is not preserved.
- Indexing and slicing not allowed.



Set Methods

The following is a list of all the methods for a **Set** object:

- **add():** Adds an element to the set.
- **update():** Update the set with the union of this set and others.
- **pop():** Removes an element from the set.
- **remove(x):** Removes the specified element.

Set Methods

- **discard(x)**: Remove the specified item.
- **clear()**: Removes all the elements from the set.
- **len()**: Prints the length of the set.
- **min()**: Searches for the minimum element in the set and prints it as the output.
- **max()**: Finds the maximum element in the set and prints it as the output.

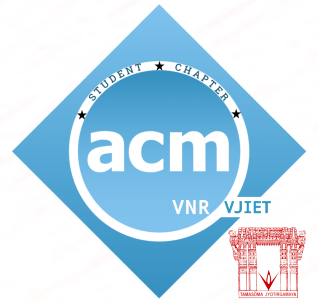
Set Methods

- **copy():** Returns a copy of the set.
- **union():** Return a set containing the union of sets.
- **intersection():** Returns a set, that is the intersection of two other sets.
- **difference():** Returns a set containing the difference between two or more sets.

Set Methods

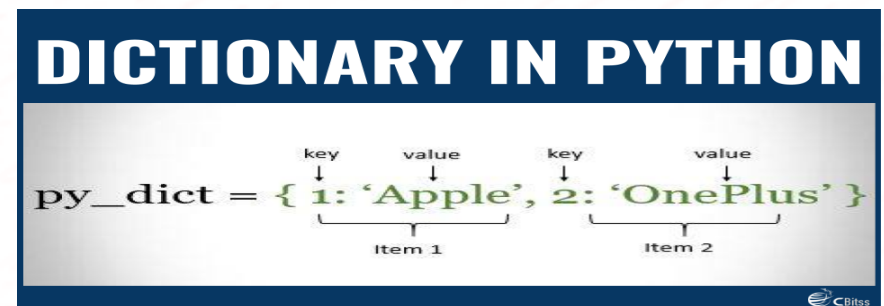
- **isdisjoint()**: Returns whether two sets have a intersection or not.
- **issubset()**: Returns whether another set contains this set or not.
- **issuperset()**: Returns whether this set contains another set or not.
- **symmetric_difference()**: Returns a set with the symmetric differences of two sets.

Advanced Datatypes – Dictionary



Dictionary:

- A Dictionary in Python is a set of key-value pairs, where the keys are unique values.
- A pair of braces `{}` creates an empty Dictionary.
- It contains values, where each value is associated with a certain key.
- Keys are immutable.



Dictionary Methods

The following is a list of all the methods for a **Dictionary** object:

- **update():** Updates the dictionary with the specified key-value pairs.
- **get():** Returns the value of the specified key.
- **items():** Returns a list containing a tuple for each key value pair.
- **keys():** Returns a list containing the dictionary's keys.

Dictionary Methods

- **values():** Returns a list of all the values in the dictionary.
- **pop():** Removes the element with the specified key.
- **popitem():** Removes the last inserted key-value pair.
- **clear():** Removes all the elements from the dictionary.
- **len():** Prints the length of the dictionary.

Dictionary Methods

- **max():** Finds the maximum element in the set and prints its key value as the output.
- **min():** Searches for the minimum element in the set and prints its key value as the output.
- **copy():** Returns a copy of the dictionary.
- **fromkeys():** Returns a dictionary with the specified keys and value.
- **setdefault():** Returns the value of the specified key. If the key does not exist: insert the key, with the specified value.



Python Programming

DAY 3

Conditional Statements

Loops

Functions

Conditional Statements

- **Conditional Statements** in Python perform different computations or actions depending on whether a specific Boolean constraint evaluates to true or false.

The basic Conditional Statements used in Python are:

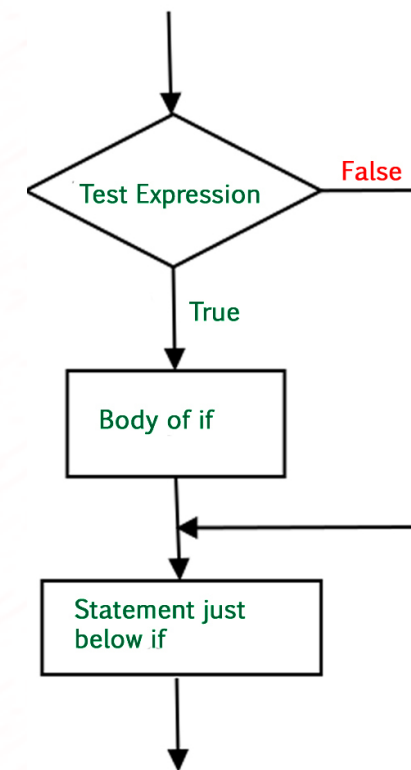
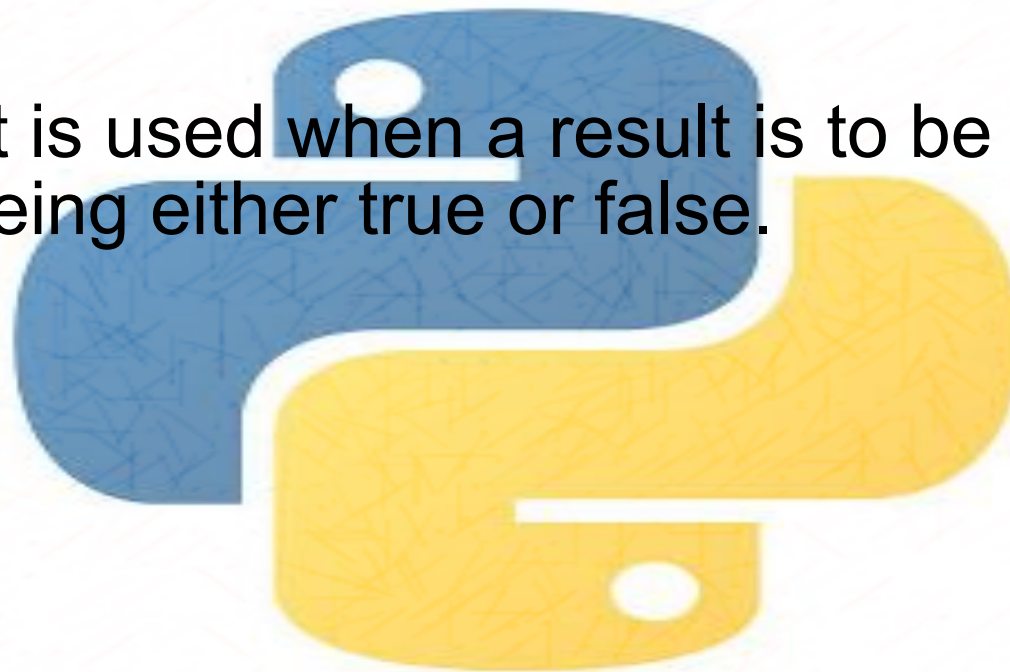
- If
- If-Else
- Elif

Conditional Statements - If

- **If statement:** It is used when a result is to be printed based on the condition being either true or false.

Syntax:

if expression:
 statement(s)



Note: Indentation is mandatory in the syntax

Conditional Statements - If

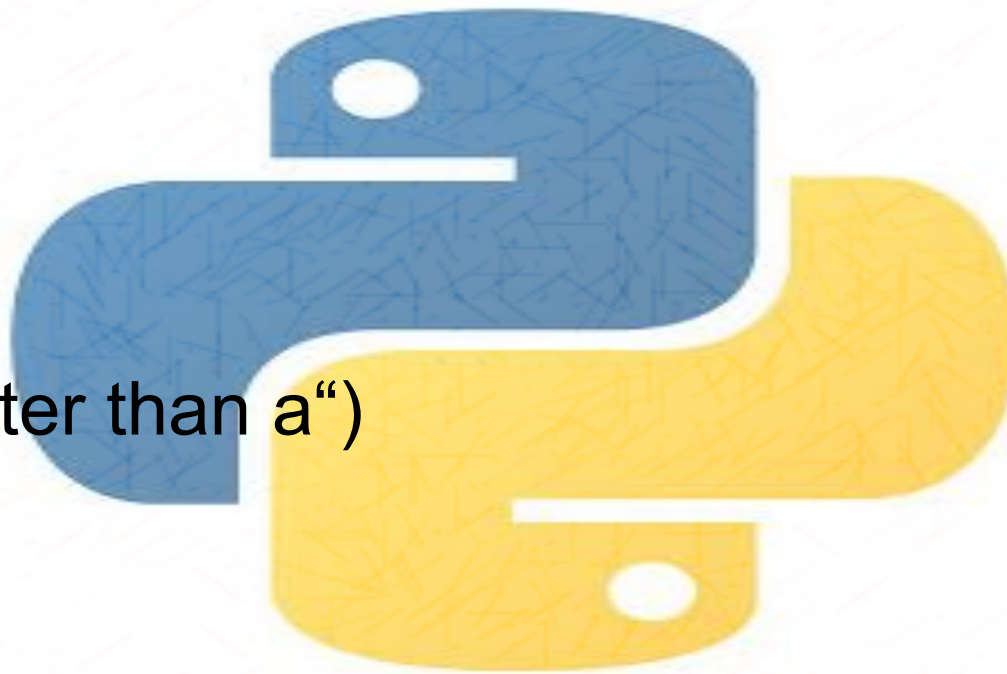
Example 1:

```
a = 33
```

```
b = 200
```

```
if b > a:
```

```
    print("b is greater than a")
```



Example 2:

```
num = int(input())
```

```
if num > 0:
```

```
    print(num, "is a positive number.")
```


Conditional Statements – Usage of Pass

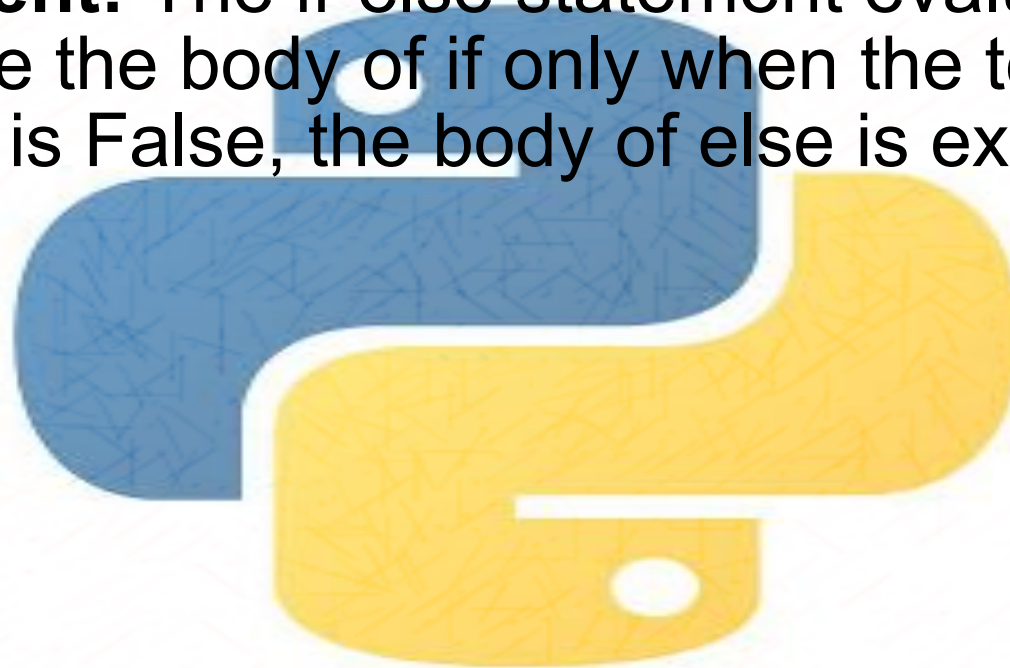
- **Pass:** if statements **cannot be empty**, but if you for some reason have an if statement with no content, put in the pass statement to avoid getting an error.

Example:

```
a = 33  
b = 200  
if b > a:  
    pass
```

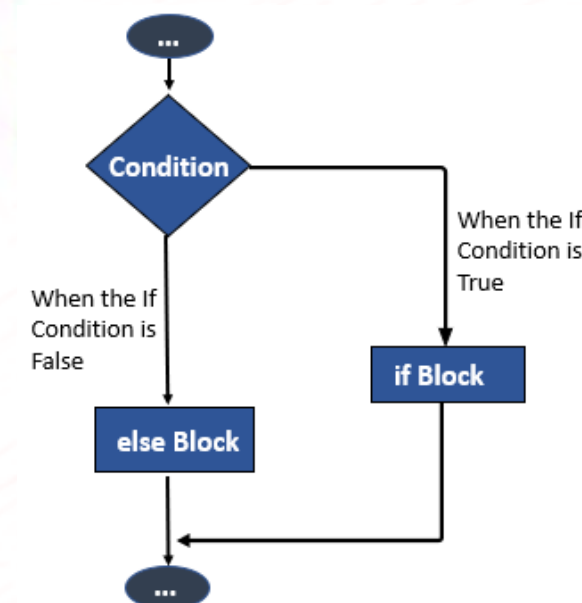
Conditional Statements – If-Else

- **If-Else statement:** The if-else statement evaluates test expression and will execute the body of if only when the test condition is True. If the condition is False, the body of else is executed.



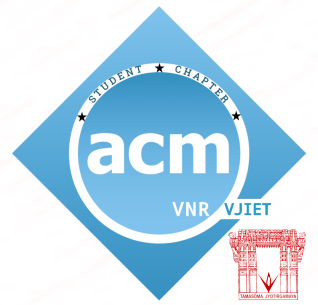
Syntax:

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



Note: Indentation is mandatory in the syntax

Conditional Statements - If-Else



Example:

```
a = 200
```

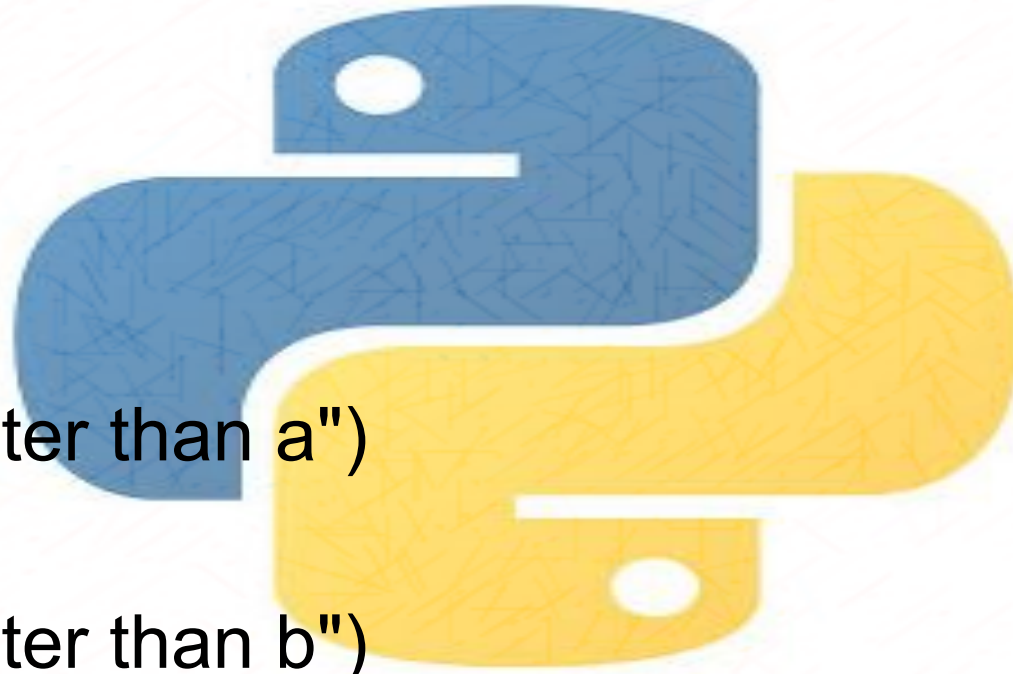
```
b = 33
```

```
if b > a:
```

```
    print("b is greater than a")
```

```
else:
```

```
    print("a is greater than b")
```



Conditional Statements – Elif

- **Elif statement:** Elif is short for else if. It allows us to check for multiple expressions. If the condition for if is False, it checks the condition of the next elif block and so on.

Syntax:

```
if expression:  
    statement(s)  
elif expression:  
    statement(s)  
else:  
    statement(s)
```

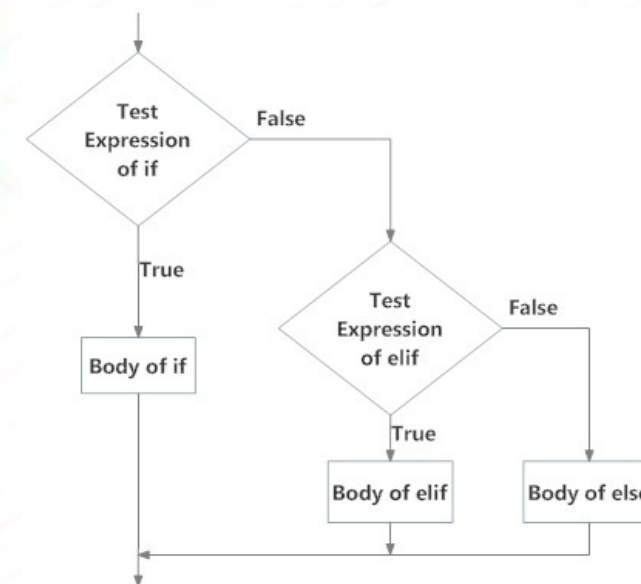
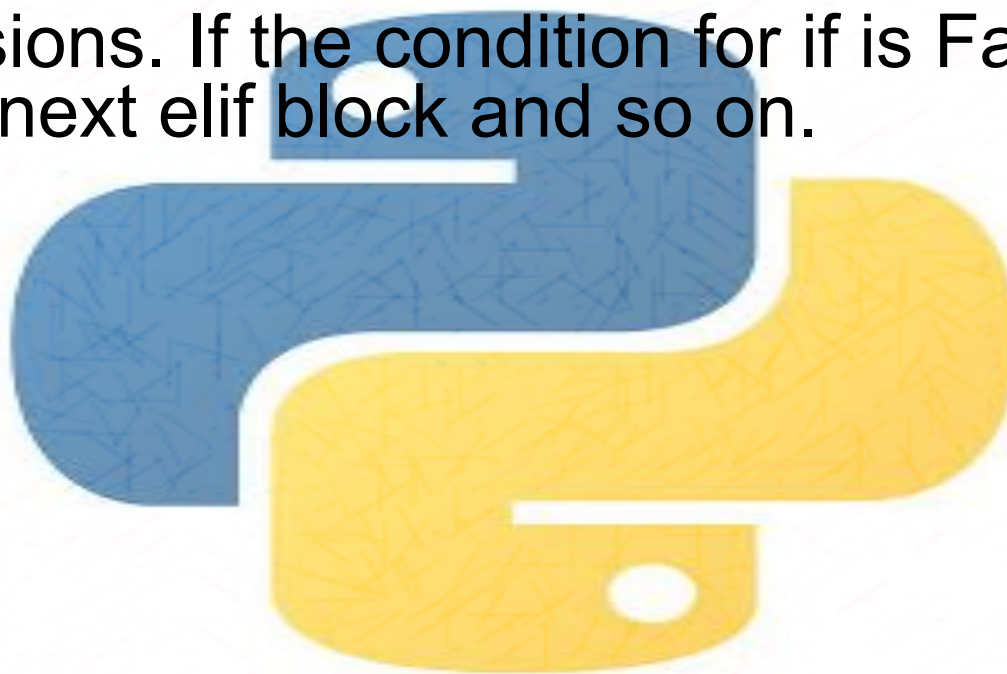


Fig: Operation of if...elif...else statement

Note: Indentation is mandatory in the syntax

Conditional Statements - Elif

Example:

```
num = input()
```

```
if num > 0:
```

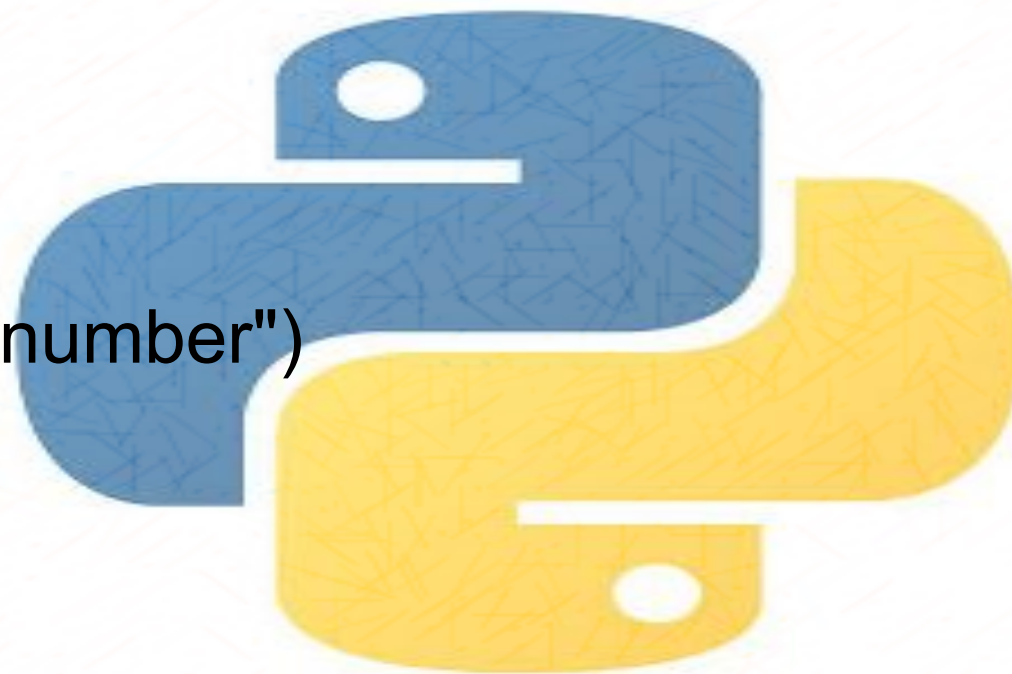
```
    print("Positive number")
```

```
elif num == 0:
```

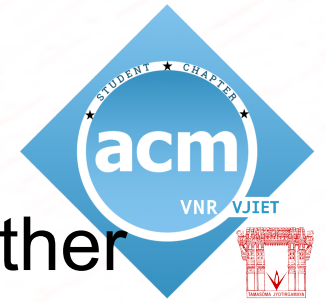
```
    print("Zero")
```

```
else:
```

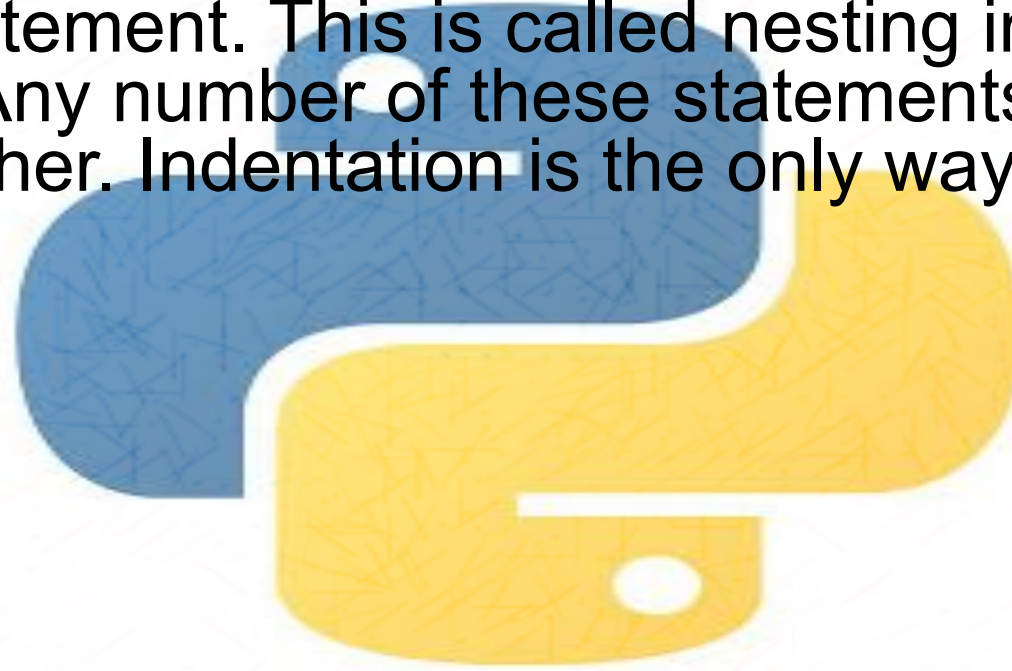
```
    print("Negative number")
```



Conditional Statements – Nesting



- **Nesting:** 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.



Syntax:

if expression:

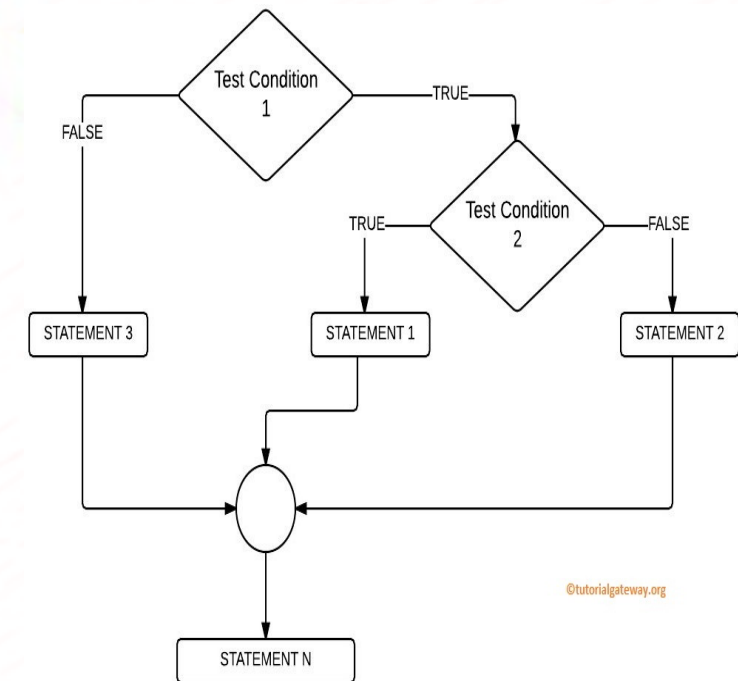
 statement(s)

if expression:

 statement(s)

else:

 statement(s)



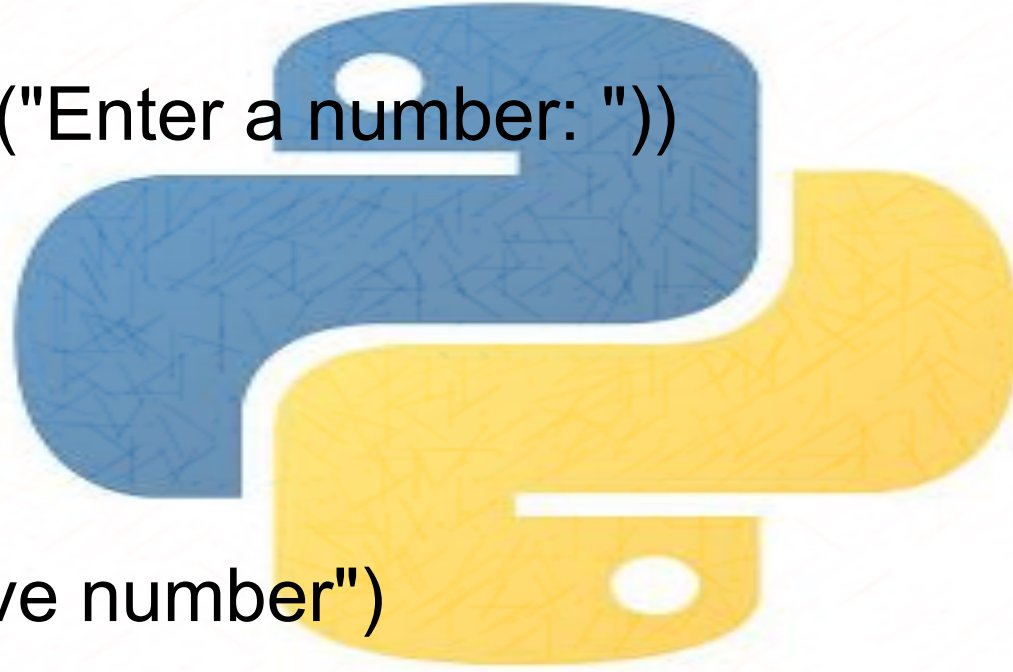
©tutorialgateway.org

Conditional Statements - Nesting



Example:

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



Loops

- A loop statement allows us to execute a statement or group of statements multiple times.

Python provides the following types of loops:

- While Loop
- For Loop

Loops – While Loop

- **While Loop:** With the while loop, it is possible to execute a set of statements as long as a condition is true.

Syntax:

```
while expression:  
    statement(s)  
    increment_stmt
```

Note: Indentation is mandatory in the syntax

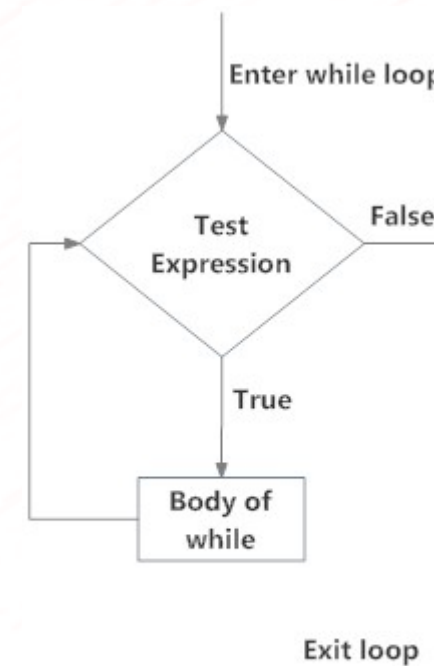
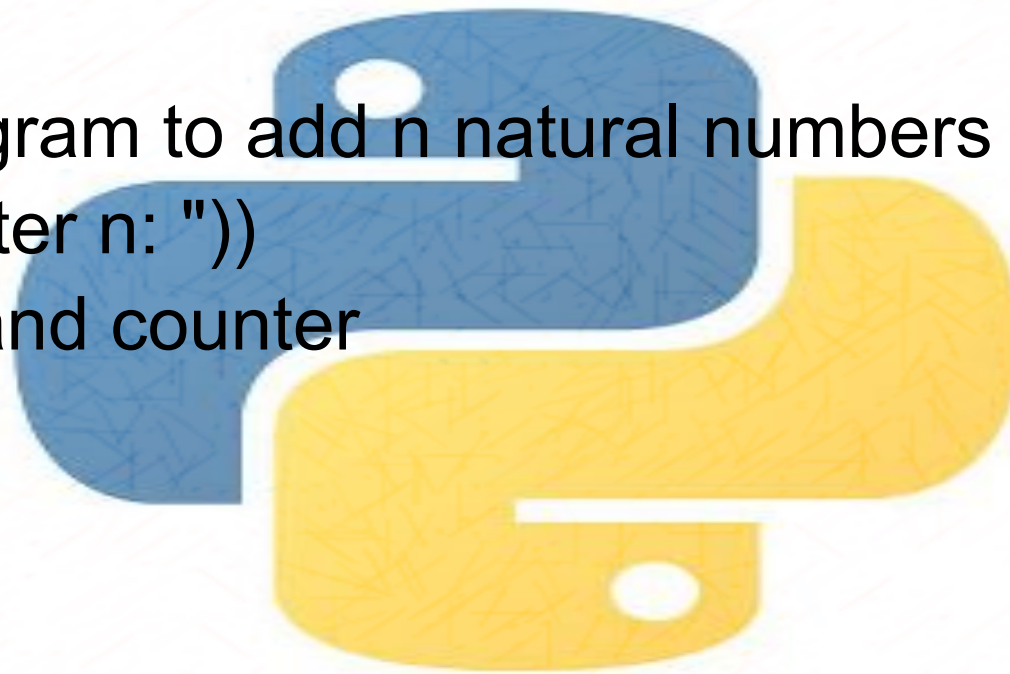


Fig: operation of while loop

Loops – While Loop

- **Example:** Program to add n natural numbers

```
n = int(input("Enter n: "))  
# initialise sum and counter  
sum = 0  
i = 1  
while i <= n:  
    sum = sum + i  
    i = i+1    # update counter  
print("The sum is", sum)
```



Loops – For Loop

- **For Loop:** A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string). This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

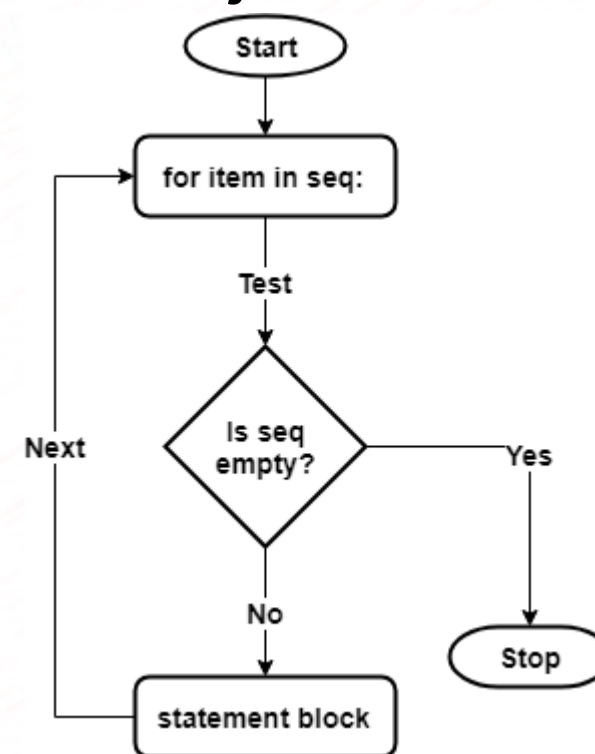
Syntax:

for val in sequence:

Body of for

Notes:

1. Indentation is mandatory in the syntax
2. For loops cannot be empty. If a For loop is empty, Include the pass statement within it.



Loops – For Loop

- **Example 1:** Print each item in a list

```
languages= ["C", "Python", "Java"]
```

```
for x in languages:
```

```
    print(x)
```

- **Example 2:** Loop through the letters in the word “Python”

```
for x in “Python”:
```

```
    print(x)
```


Loops – Nesting

- **Nested Loops:** A nested loop is a loop inside a loop. The "inner loop" will be executed one time for each iteration of the "outer loop"

Example: Print each adjective for every web

```
adj = ["red", "big", "tasty"]
```

```
web = ["HTML", "CSS", "Jquery"]
```

```
for x in adj:
```

```
    for y in web:
```

```
        print(x, y)
```

Loops – Break Statement

- **Break:** With the break statement we can stop the loop even if the while condition is true

Example: Exit the loop when i is 3

```
i = 1
```

```
while i < 6:
```

```
    print(i)
```

```
    if i == 3:
```

```
        break
```

```
    i += 1
```



Loops – Continue Statement

- **Continue:** With the continue statement we can stop the current iteration, and continue with the next.

Example: Continue to the next iteration if i is 3

```
i = 1
```

```
while i < 6:
```

```
    print(i)
```

```
    if i == 3:
```

```
        continue
```

```
    i += 1
```

Loops – Range() Function

- **The Range() Function:** To loop through a set of code a specified number of times, we can use the range() function. The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

Example 1:

```
for x in range(6):  
    print(x)
```


Loops – Range() Function

- The range() function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: range(2, 6), which means values from 2 to 6 (but not including 6)

Example 2:

```
for x in range(2, 6):  
    print(x)
```

Loops – Range() Function

- The range() function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter: range(2, 30, 3)

Example 3:

```
for x in range(2, 30, 3):  
    print(x)
```

Functions

- **Functions** are a convenient way to divide your code into useful blocks, allowing us to order our code, make it more readable, reuse it and save some time. There exist two types of Functions in Python, User Defined Functions and Pre-Defined Functions.
- User Defined Functions in python are defined using the block keyword "def", followed with the function's name as the block's name.

Syntax:

```
def fun_name(argument1 , argument2, ...):  
    body of the function
```

Functions

- **Example 1:** Simple Function, without arguments

```
def my_function():  
    print("Welcome to python workshop!")
```

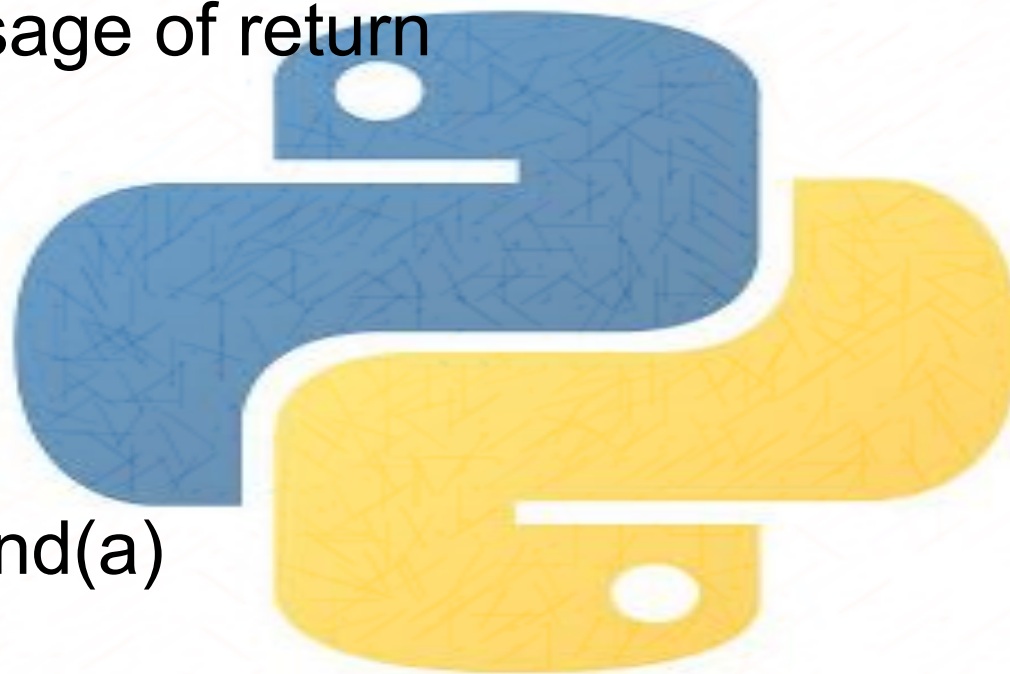
Example 2: Function with arguments

```
def my_function_with_args(username, greeting):  
    print("Hello, %s , Welcome !, I wish you %s"%(username,  
greeting))  
my_function_with_args("Sudha", "to have a great day!")
```


Functions

- **Example 3:** Usage of return

```
def fib(n):  
    result = []  
    a,b=0,1  
    while a<n:  
        result.append(a)  
        a,b=b,a+b  
    return result
```



Note: The return statement returns with a value from a function. return without an expression argument returns None. Falling off the end of a function also returns None.

Functions

- **Example 4:** Defining Default Argument Values

```
def compute(x,y=0,itr=2):
```

```
    sum=0
```

```
    while itr>0:
```

```
        sum=sum+x+y
```

```
        itr-=1
```

```
    return sum
```

The above function can now be called as:

```
compute(1)
```

```
compute(1,2)
```

```
compute(1,2,4)
```





Thank You