

PYTHON- BASICS

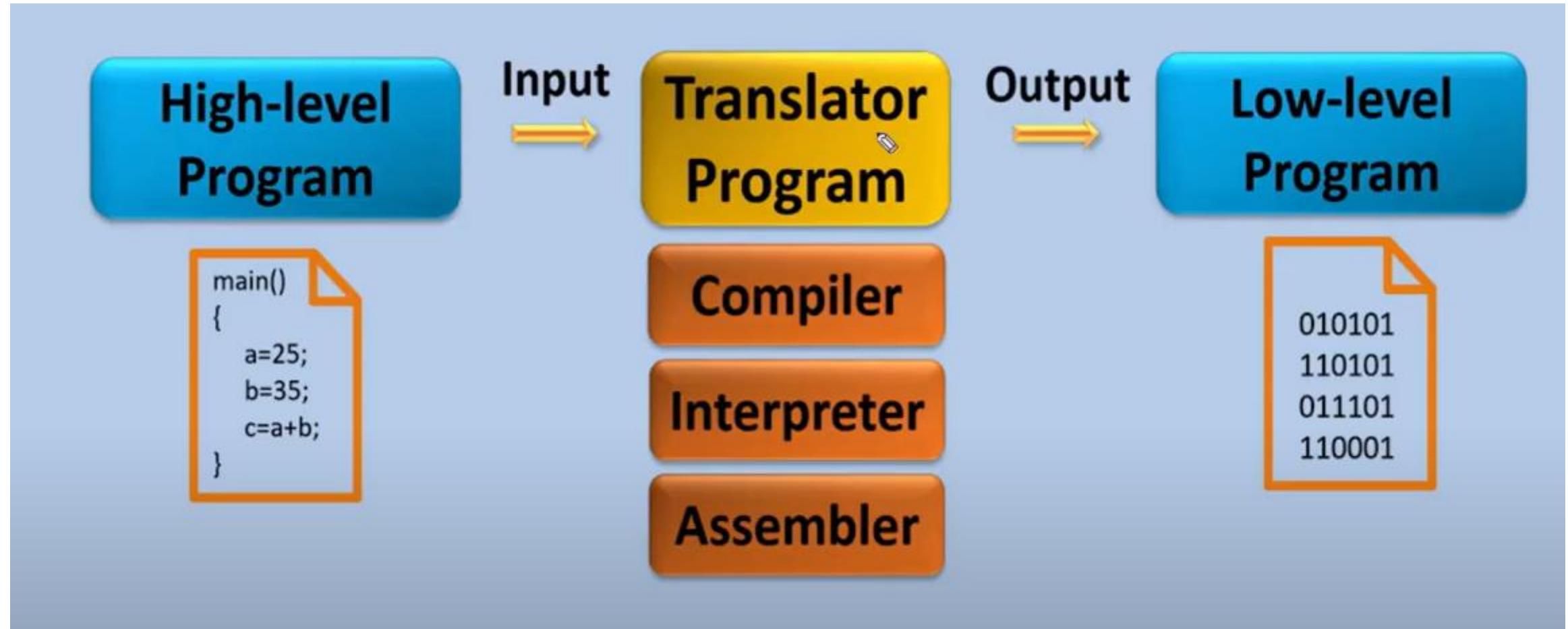
Dr Sivabalan,
Technical Training Advisor
Sivabalan.n@nttdata.com

programming languages:-

Each programming language has its own syntax and rules

1. Low-Level Languages
2. High-Level Languages

Python-Basics



What is Python?

Python is a high level programming language like C,C++

Python is a simple, high level, interpreted, general purpose, dynamically typed and object oriented programming language created by Guido Van Rossum in 1991.

Python is simple

- Python is easy to use.
- python's programs are simple to write and hence it is easy to learn.
- Lots of built-in modules, packages and Frameworks

Python-Basics

Hello World

Java:

```
// Hello World in Java
class HelloWorld {
    static public void main(String args[]) {
        System.out.println("Hello World!");
    }
}
```

C++:

```
// Hello World in C++
#include <iostream.h>
Main() {
    cout << "Hello World!" << endl;
    return 0;
}
```

Python:

```
# Hello World in Python
print("Hello World!")
```

Python combines remarkable power with very clean, simple, and compact syntax.

Python is high-level

- More user-friendly
- There is automatic memory management
- Rich set of libraries and functions ex. len()

Python is dynamically typed

- Like C or C++, we don't need to specify datatypes of identifiers.
- Python evaluates datatypes at runtime

Python is general purpose programming language:-

Python is used for web development, machine learning, artificial intelligence, data analysis, data science, web scraping, scripting, scientific computing, software dev etc

Applications of Python

Software Applications

Web Development

Data Science

Data Analysis

Machine Learning

Artificial Intelligence

Web scraping

Image Processing

Game Development

Scripting

Network Programming

Scientific & Numeric

Console & GUI based apps

Python-Basics

Web applications :- Django, flask, pyramid Frameworks

GUI application:- libraries like Kivy, Tkinter

Scientific and numeric applications:- Scipy, pandas, numpy

Data science:- numpy, pandas, seaborn, matplotlib

Image processing:- OpenCv library

Game Development:- Pygame library

- What are variables and need of variables ?
- How to create variables & working of variables ?
- Rules for creating variables (Naming conventions)
- do's & dont's
- Keywords in python

How variable works in C ?

```
int a = 20;    int b = 20;    int c = a;    int d = 30;
```

a

20

1244

b

20

1248

c

20

1252

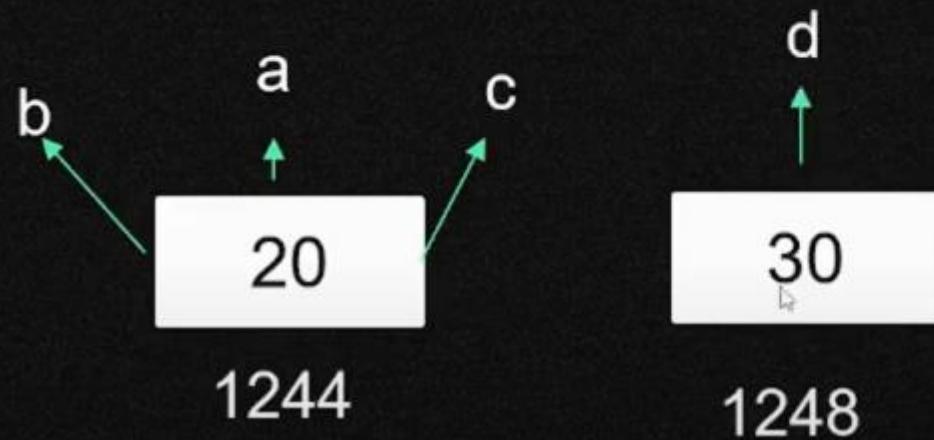
d

30

1256

How variable works in python ?

```
a = 20;           b = 20;           c = a;           d = 30;
```



How variable works in python ?

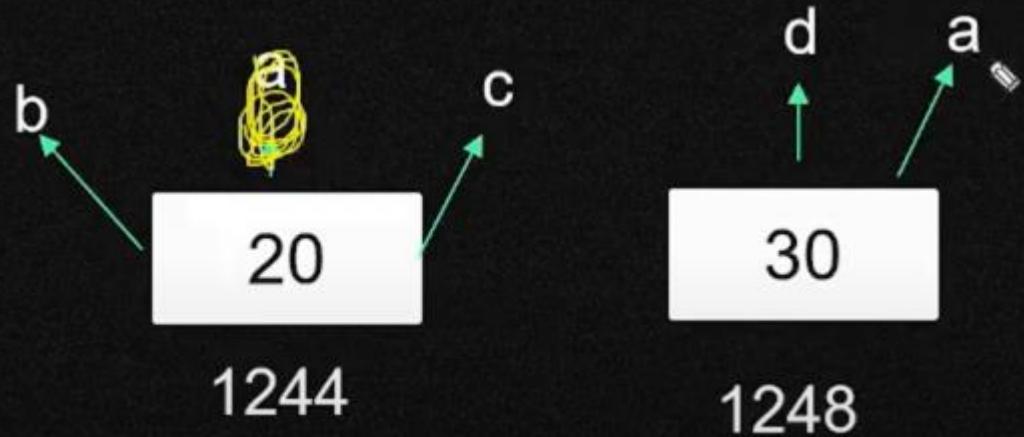
```
a = 20;
```

```
b = 20;
```

```
c = a;
```

```
d = 30;
```

```
a = d;
```



How variable works in python ?

a = 20;

b = 20;

c = a;

d = 30;

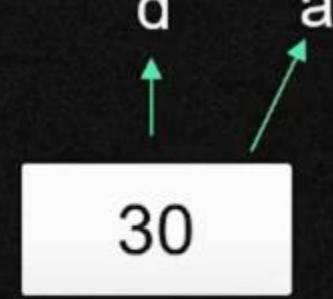
a = d;

b = 40;



20

1244



30

1248



40

1252

How variable works in python ?

```
a = 20;
```

```
b = 20;
```

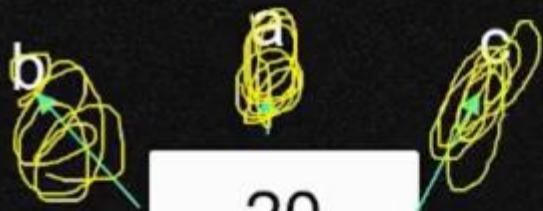
```
c = a;
```

```
d = 30;
```

```
a = d;
```

```
b = 40;
```

```
c = b;
```



1244

1248

1252

How variable works in python ?

```
a = 20;
```

```
a = d;
```



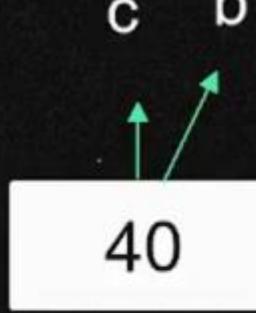
```
b = 20;
```

```
b = 40;
```



```
c = a;
```

```
c = b;
```



```
d = 30;
```

Garbage Collector

Difference between Java/c variables and python variables?

c/java allocates memory for variables.

python allocates memory for values and not for variables.

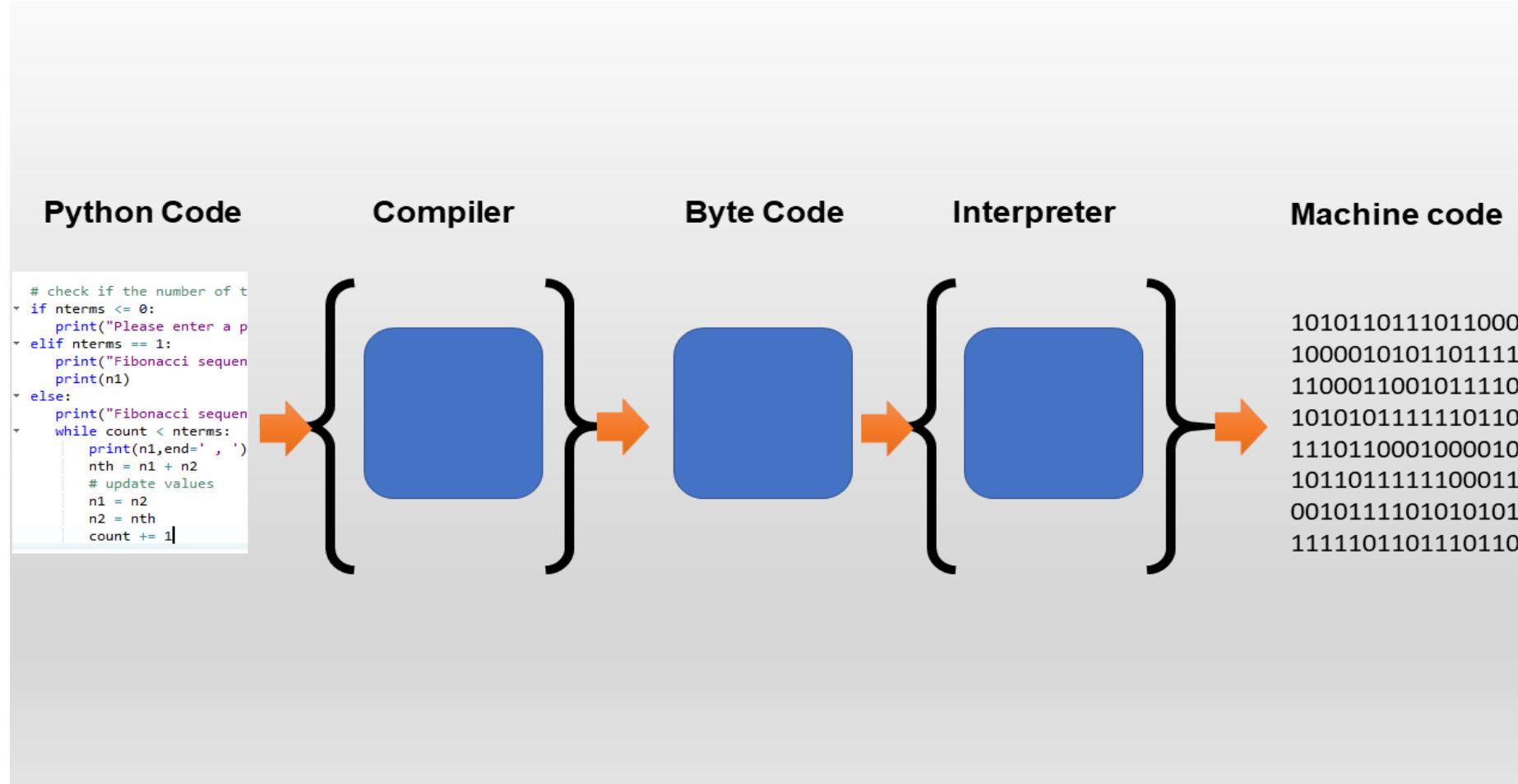
Python has efficient memory management.

Keywords are **special words** in any programming language.

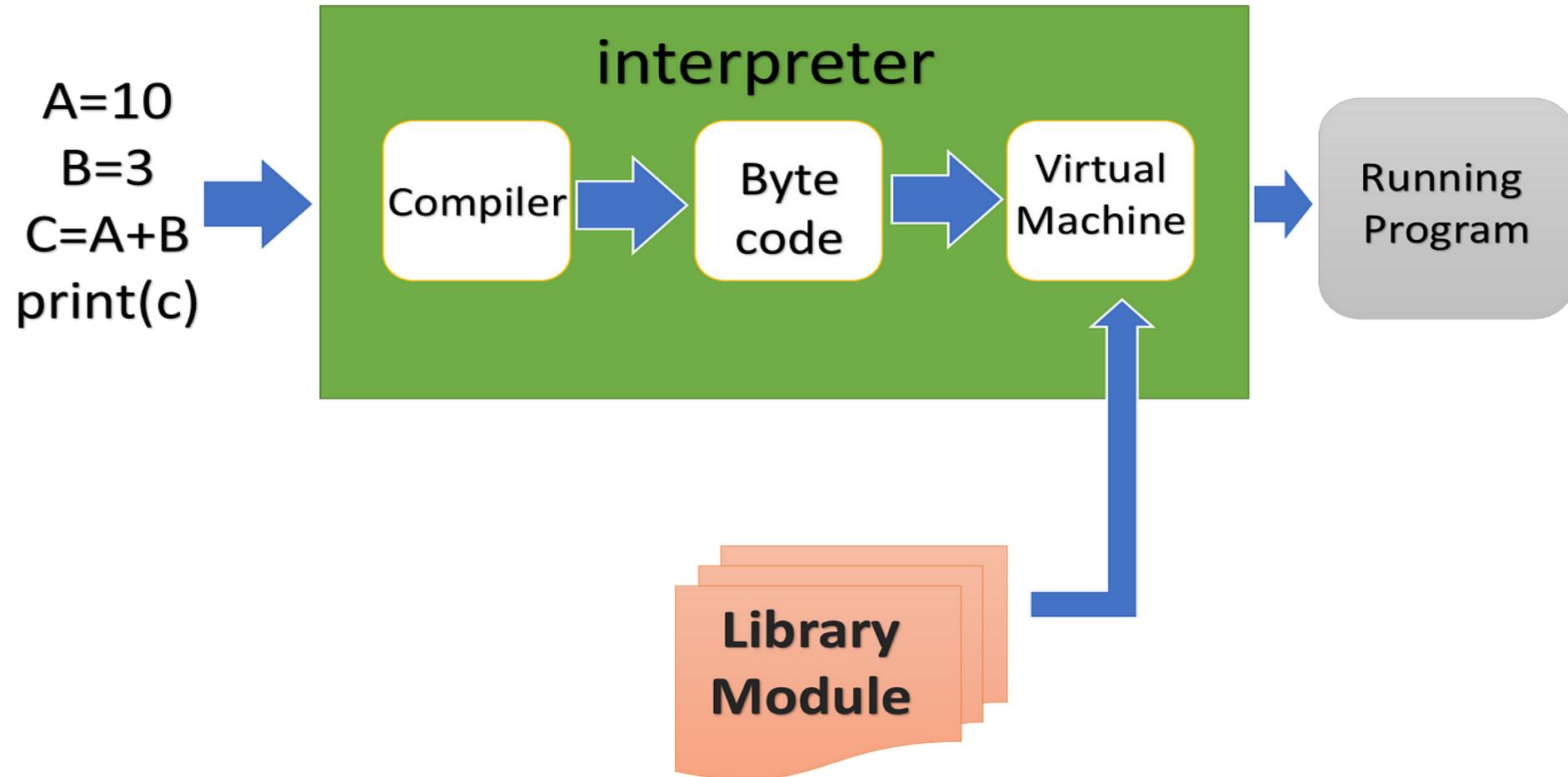
These are **reserved words** in python

Every keyword has specific meaning

Python-Basics



Python-Basics



Comments in Python

An important part of programming

Need of comments

helps you and others to understand later on the intention of your code.

This allows you to more easily find errors, to fix them, to improve the code later on, and to reuse it in other applications as well.

Datatypes in Python

- **type()** function
- **isinstance()** function

Below are datatypes available in python:-

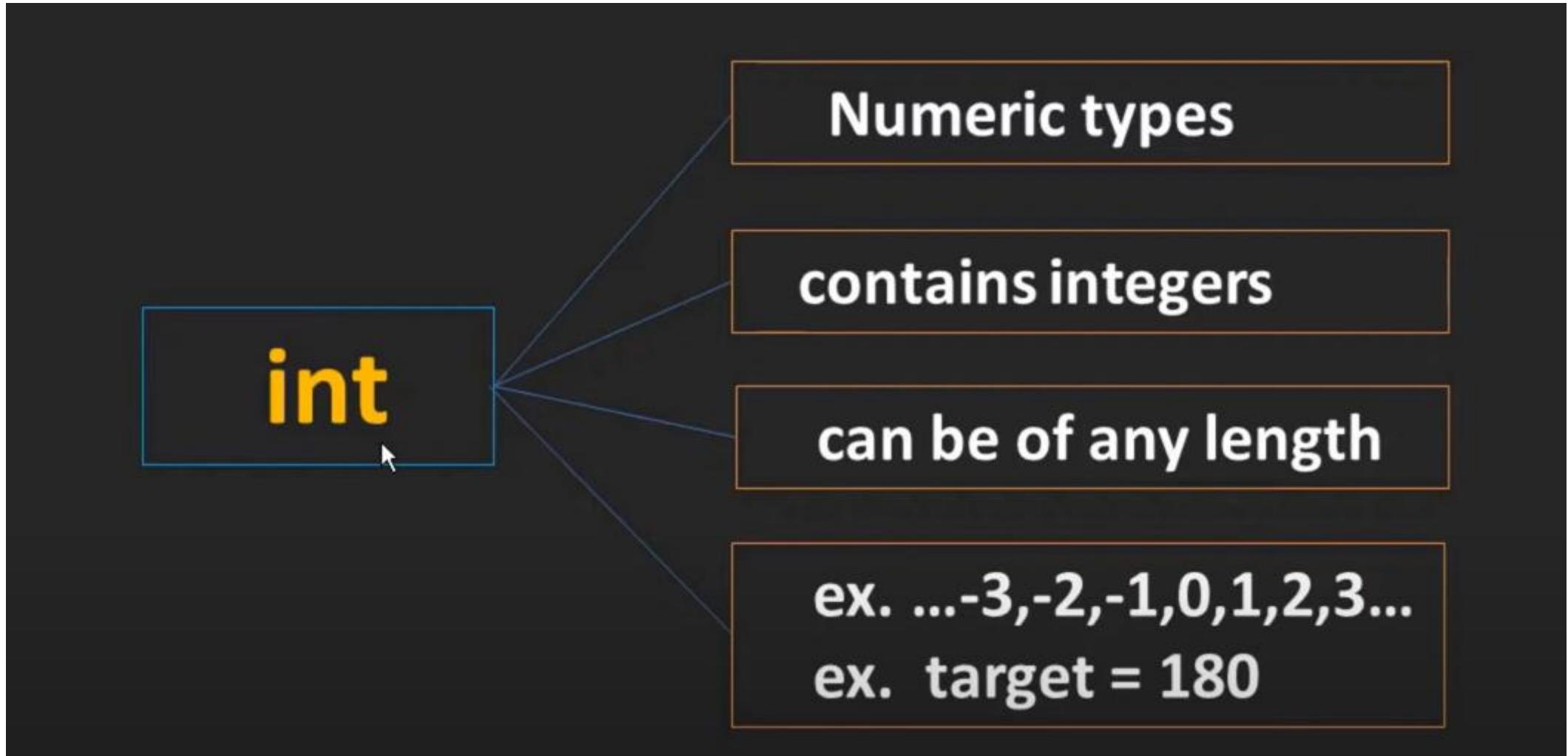
- Numeric types:- int, float, complex (special type)
- Sequence types:- str, list, tuple
- Mapping types:- dict
- Set types:- set, frozenset
- Boolean Type:- Bool
- Binary Types:- bytes, bytearray
- None type:- NoneType

Numeric - types

int

float

complex



float

Numeric type

floating point numbers.

accurate upto 15 decimal places

ex. 3.2, 4.0, 8.9,-12.4

ex. temperature = 37.4

complex

written in form of $a+bj$

‘a’ is real part.can be integer or float.

‘bj’ is an imaginary part.can be integer or float.

Ex. $c = 5+4j$

string

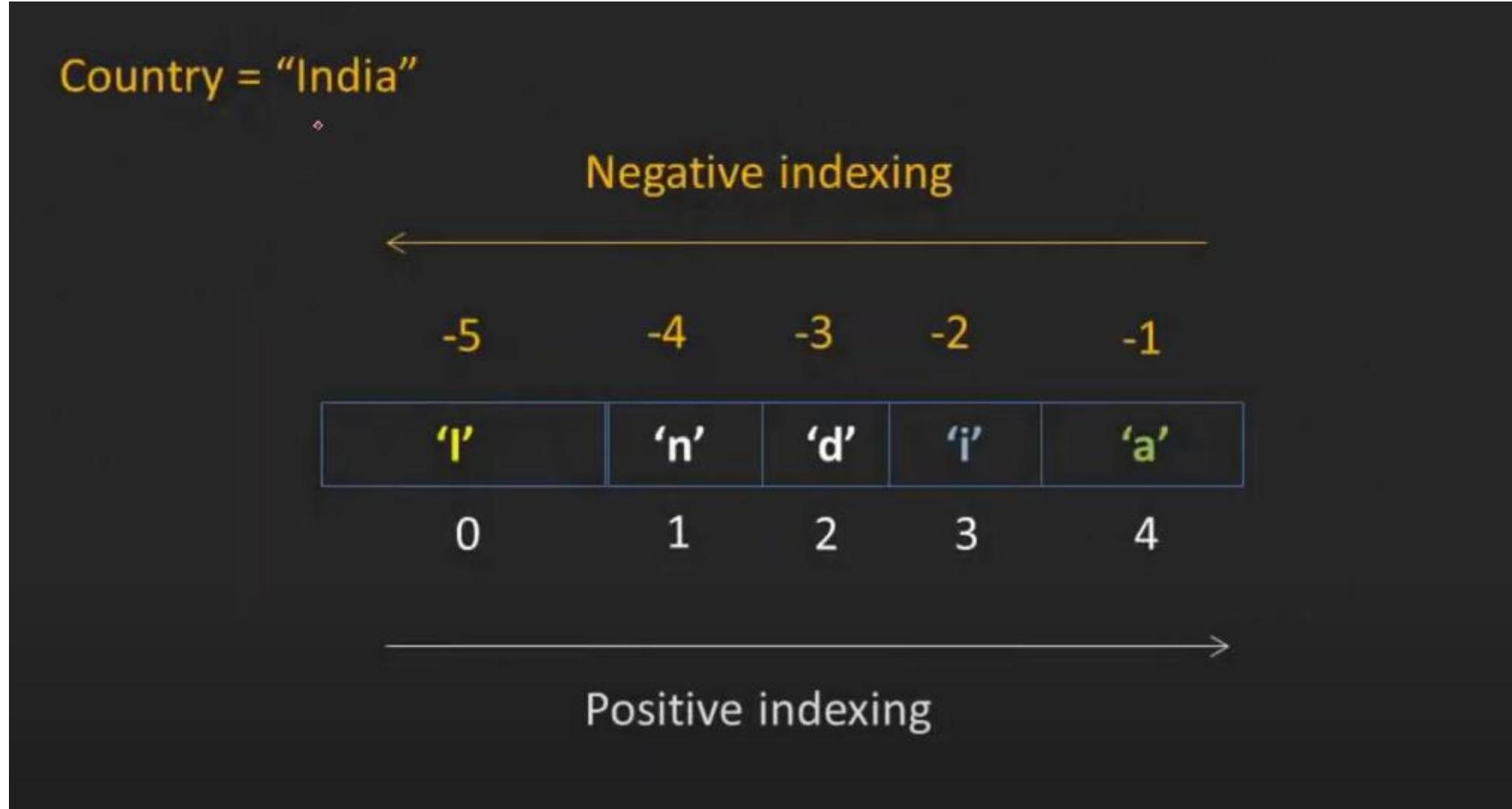
Sequence of characters.

Anything enclosed in quotes.

triple quotes are used for multi-line strings.

Ex. name = “hello”

Python-Basics



sequence - types

list

tuple

dictionary

Set,frozenset

list

list of data's having different data types.

Ordered mutable sequence of items.

Items separated by commas enclosed in [].

Python-Basics

```
My_list = ['facebook', 12.5 ,300, 2+3j , [1,2,3] ]
```

Negative indexing

←

-5 -4 -3 -2 -1

'facebook'	12.5	300	2+3j	[1,2,3]
------------	------	-----	------	---------

0 1 2 3 4

→

Positive indexing

tuple

Collection of different data types.

Immutable, ordered

Items are separated with comma and enclosed in ().

Ex. t = (12,10.5,'hello')

dictionary

Unordered set of key-value paires.

Key has primitive datatype and value has any datatype.

Key-value paires are separated by comma.

Ex. d = {sachin:100 , kohli:90, dhoni:80}

set

Unordered collection of unique items.

Items are written inside { } and separated by commas.

Items are not in order.

Ex. a = {1,2,3,4,"hello"}

range

It gives immutable sequence of numbers betn start and stop.

Syntax:- range(start,stop,step)

Default starting is 0 and step is 1.

Ex. range(0,11) -> 0,1,2....10

None

None datatype means an object
that doesn't contain any value.

Ex. `a = None`

bool

Two values, True and False.

Used to determine given statements are true or false.

True is non-zero and False is 0.

Ex. a = True

Q.1 what is the output of following?:-

```
print(type("35"))
```

- 1) <class 'float'>
- 2) <class 'str'>
- 3) <class 'int'>

Q.1 what is the output of following?:-

```
a = {1,2,2,3,4,3}  
print(a)
```

- 1) {1,2,3,4,3}
- 2) {1,2,2,3,4,3}
- 3) {1,2,3,4}

Q.1 what is the output of following?:-

```
print(11,12,13,14)
```

- 1) 11,12,13,14**
- 2) 11 12 13 14**
- 3) (11,12,13,14)**

Operators :- Symbols to carry out operations on
operands (values or data)

Ex. $2 + 3$

$+$ is an operator.

2,3 are operands

→ Magic methods

Below are operators in Python:-

- Arithmetic operators
- Comparison operators
- Assignment and compound operators
- Bitwise operators
- Logical operators

Special operators:-

- Membership operators
- Identity operators
- Walrus operator

Arithmetic operators are:-

- 1) '+' (for addition)
- 2) '-' (for subtraction)
- 3) '*' (for multiplication)
- 4) '/' (for division)
- 5) '%' (for remainder)
- 6) '//' (for floor division)
- 7) '**' (for exponentiation)

Q. What value will be printed after executing following statement?

```
print(18/4)
```

- A) 4.0
- B) 4.5
- C) 5
- D) 5.0

Q. What value will be printed after executing following statement?

```
print(18//4)
```

- A) 4.0
- B) 4
- C) 5
- D) 4.5

Q. What value will be printed after executing following statement?

```
print(18%4)
```

- A) 4.0
- B) 4.5
- C) 2
- D) 2.0

Q. What value will be printed after executing following statement?

```
print(18.0//4)
```

- A) 4.0
- B) 4.5
- C) 4
- D) 5.0

String in Python:-

- A string is a sequence of characters. Ex. “shantanu”
- Anything enclosed inside single quotes ,double quotes except escape sequences is [→]string.
- String is a sequence of unicode characters.

Creating string:-

‘hello’

“hello”

```
print('hello')  
print("hello")
```

Output:- hello
hello

- Triple quotes are also used but for multi-line strings and docstrings.

```
print("""hello,welcome to  
world of python""")
```

Accessing substring (characters) in string:-

Two ways:-

- 1) Indexing
- 2) Slicing

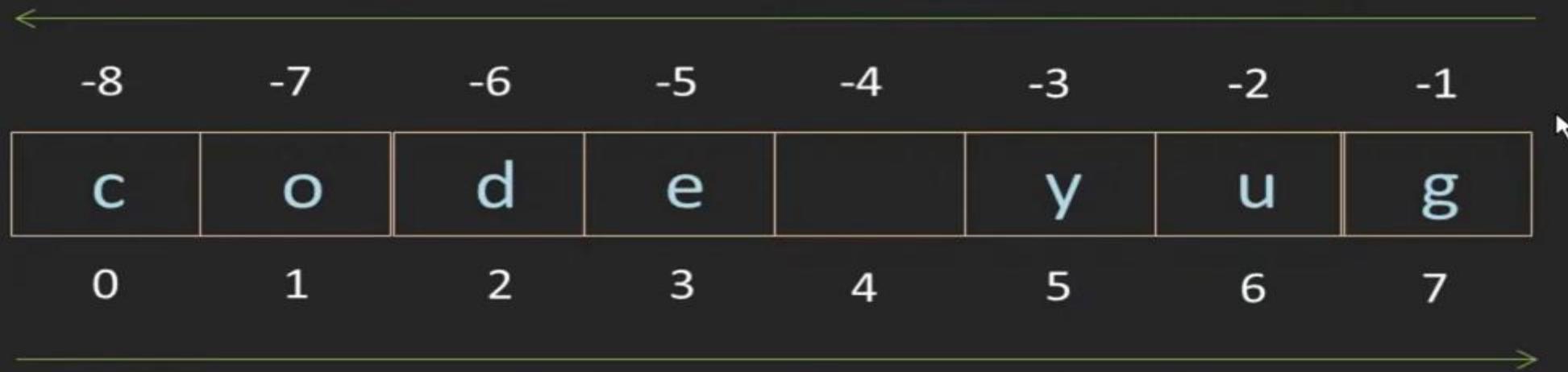
Indexing:- used to access individual(single) element of string.

slicing:- used to access group of elements of a string. We can access range of characters using slicing.

Index:-

- In python, Every element is represented by **index**.
Name = “code Yug”

Negative indexing / Reverse Indexing

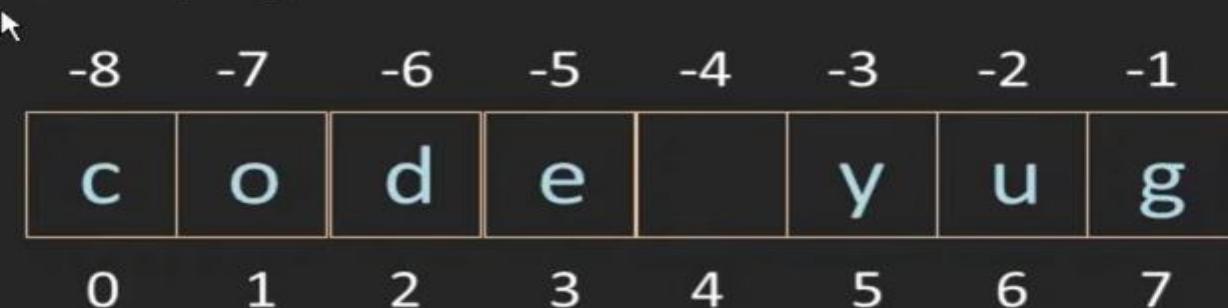


Positive indexing / Forward Indexing

Accessing Single Element:-

- To access single element, python uses indexing.

Name = “code Yug”



Syntax:- var_name[index]

Name[0] ---> c

Name[2] ---> d

Name[5] ---> y

Name[-1] ---> g

Name[-4] --->

Name[-8] ---> c

slicing:-

- To access a range of items (sub-string), we will use 'slicing'.
- You can return range of characters using slicing.
- [:] is a slice operator.

Syntax:-

Var_name[start:stop:step]

- start :- starting point of substring.
- stop :- It indicated end of string.
- step :- difference between indexing.

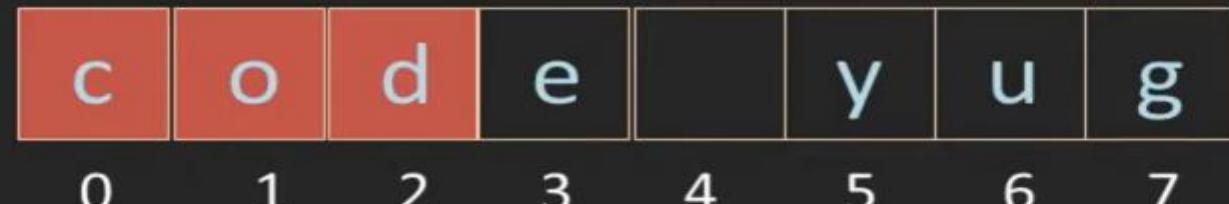
important:-

- start,stop,step are optional.
- Default start is index 0.
- Default step is 1 .
- end = stop -1
- Character at stop index is not printed.
- Positive index :- Go in forward direction
Negative index :- Go in Negative direction
- If stop not found,nothing is returned.

Python-Basics

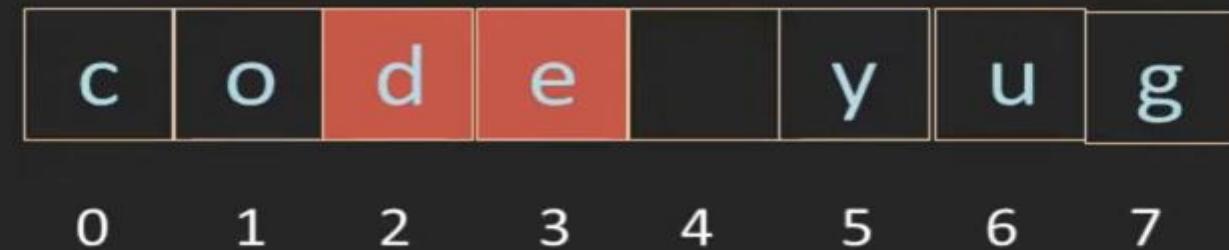
```
print(Name[0:3:1])      ----> cod
```

-8 -7 -6 -5 -4 -3 -2 -1



```
print(Name[2:4])      ----> de
```

-8 -7 -6 -5 -4 -3 -2 -1



Python-Basics

```
print(Name[0:5:2])      ----> cd_
```



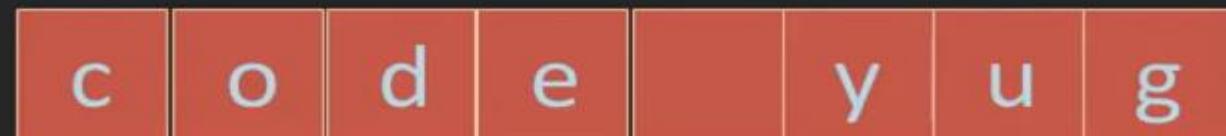
```
print(Name[:4])
```



Python-Basics

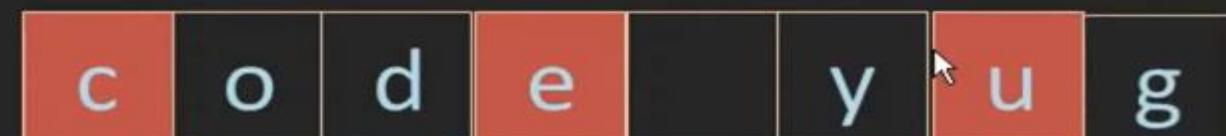
```
print(Name[:])      ----> code yug
```

-8 -7 -6 -5 -4 -3 -2 -1



```
print(Name[0::3])      ----> ceu
```

-8 -7 -6 -5 -4 -3 -2 -1



Python-Basics

```
print(Name[-1:-4:-1]) ----> guy
```

-8 -7 -6 -5 -4 -3 -2 -1



```
print(Name[-1:-5:-2]) ----> gy
```

-8 -7 -6 -5 -4 -3 -2 -1



Python-Basics

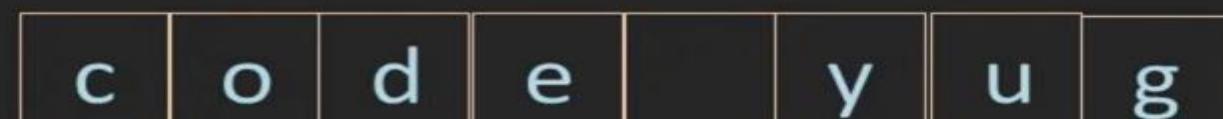
```
print(Name[-2:-6:2])
```

-8 -7 -6 -5 -4 -3 -2 -1



```
print(Name[-1: :-1])
```

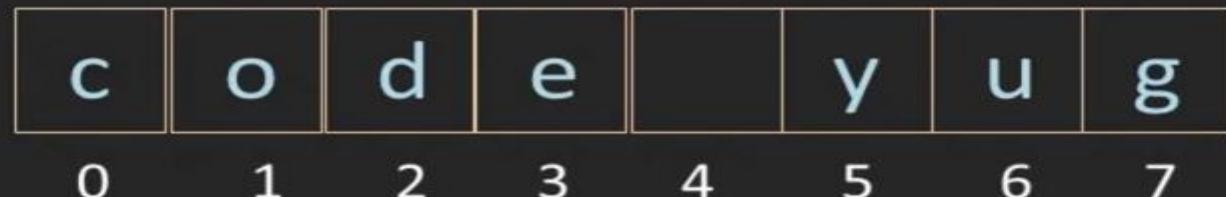
-8 -7 -6 -5 -4 -3 -2 -1



Python-Basics

```
print(Name[-2:-6:2]) ----> "
```

-8 -7 -6 -5 -4 -3 -2 -1



```
print(Name[-1: :-1]) ----> guy edoc
```

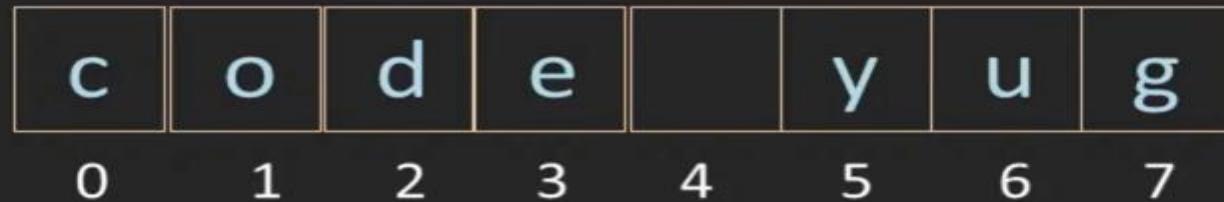
-8 -7 -6 -5 -4 -3 -2 -1



Python-Basics

```
print(Name[-3:-11]) ----> "
```

-8 -7 -6 -5 -4 -3 -2 -1



```
print(Name[-1:-1:-1]) ----> "
```

-8 -7 -6 -5 -4 -3 -2 -1



```
string = "Hello, world!"  
print(string[7:12])  
a) world b) world! c) , worl d) , world
```

```
string = "Hello, world!"  
print(string[7:12])  
a) world b) world! c) , worl d) , world
```

```
string = "Hello, world!"  
print(string[:5])  
a) Hello b) Hello, c) , world! d) world!
```

```
string = "Hello, world!"  
print(string[:5])  
a) Hello b) Hello, c) , world! d) world!
```

```
string = "Hello, world!"  
print(string[-6:-1])
```

- a) world
- b) world!
- c) , worl
- d) , world

```
string = "Hello, world!"  
print(string[-6:-1])
```

- a) world
- b) world!
- c) , worl
- d) , world

Which of the following is the correct way to slice a string to get the last 3 characters?

- a) `string[-3:]`
- b) `string[3:]`
- c) `string[:-3]`
- d) `string[-3:-1]`

Which of the following is the correct way to slice a string to get the last 3 characters?

- a) `string[-3:]`
- b) `string[3:]`
- c) `string[:-3]`
- d) `string[-3:-1]`

Python-Basics- Q5

```
string = "Hello, world!"  
print(string[-5:2:-2])
```

- a) ol
- b) ow
- c) ,ol
- d) dlro

Python-Basics- Q6

```
string = "Python is fun!"  
print(string[::-2][::-1])
```

Which of the following slices would extract the substring "world" from the string "Hello, world!"?

- a) `string[7:12]`
- b) `string[-6:-1]`
- c) `string[7:-1]`
- d) All of the above

```
string = "Programming"
```

```
print(string[3:len(string) - 4])
```

```
string = "Programming"
```

```
print(string[3:len(string) - 4])
```

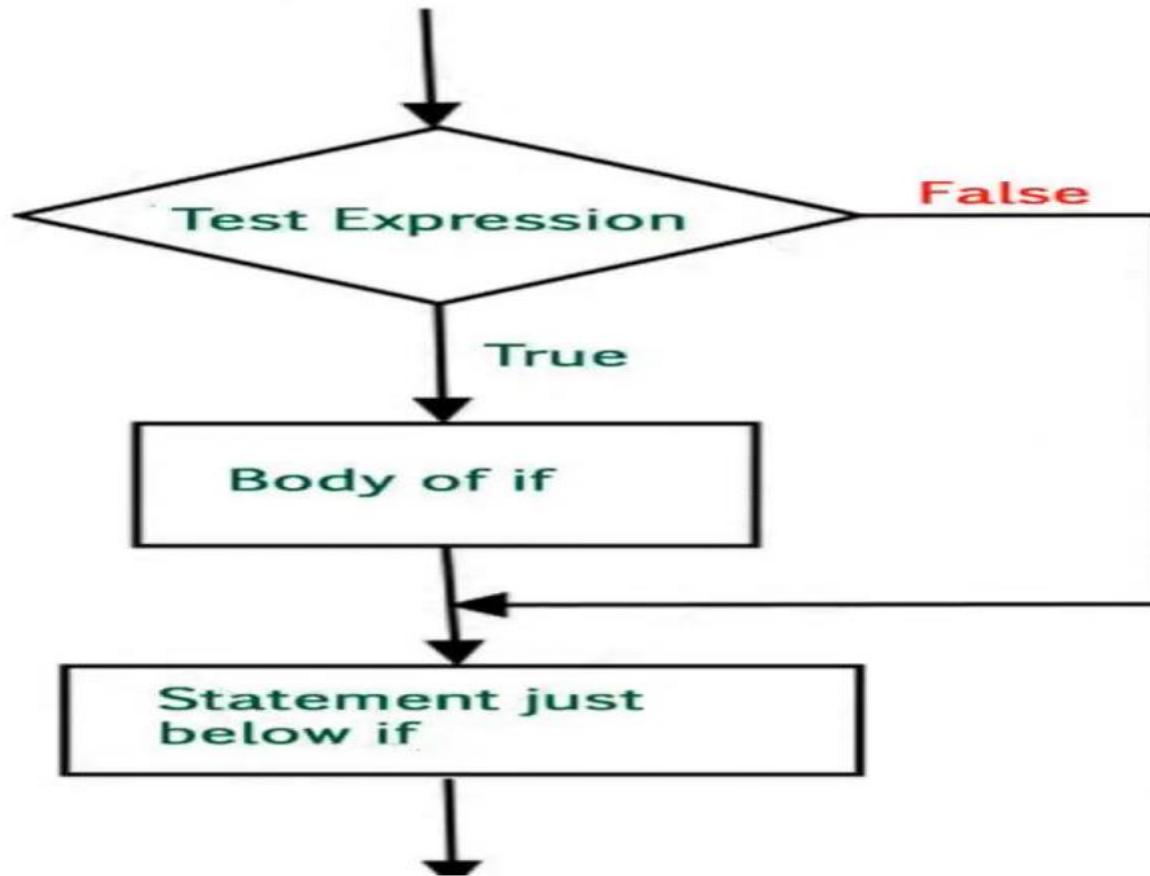
if statement:-

- If statement is used for decision making.
- In if statement, we check a condition. depending on truth value of condition, decision will be taken.
- Written using 'if' keyword.

Syntax:-

```
if condition/test_expression:  
    statement 1  
    statement 2  
    #rest of statements
```

Python-Basics- Decision Making



if-else in python:-

- The if-else statement provides an else block combined with if block which is executed in the False case of condition.

- Syntax:-

```
if test_expression:  
    body of if  
else:  
    body of else
```

True:- if block
False:- else block

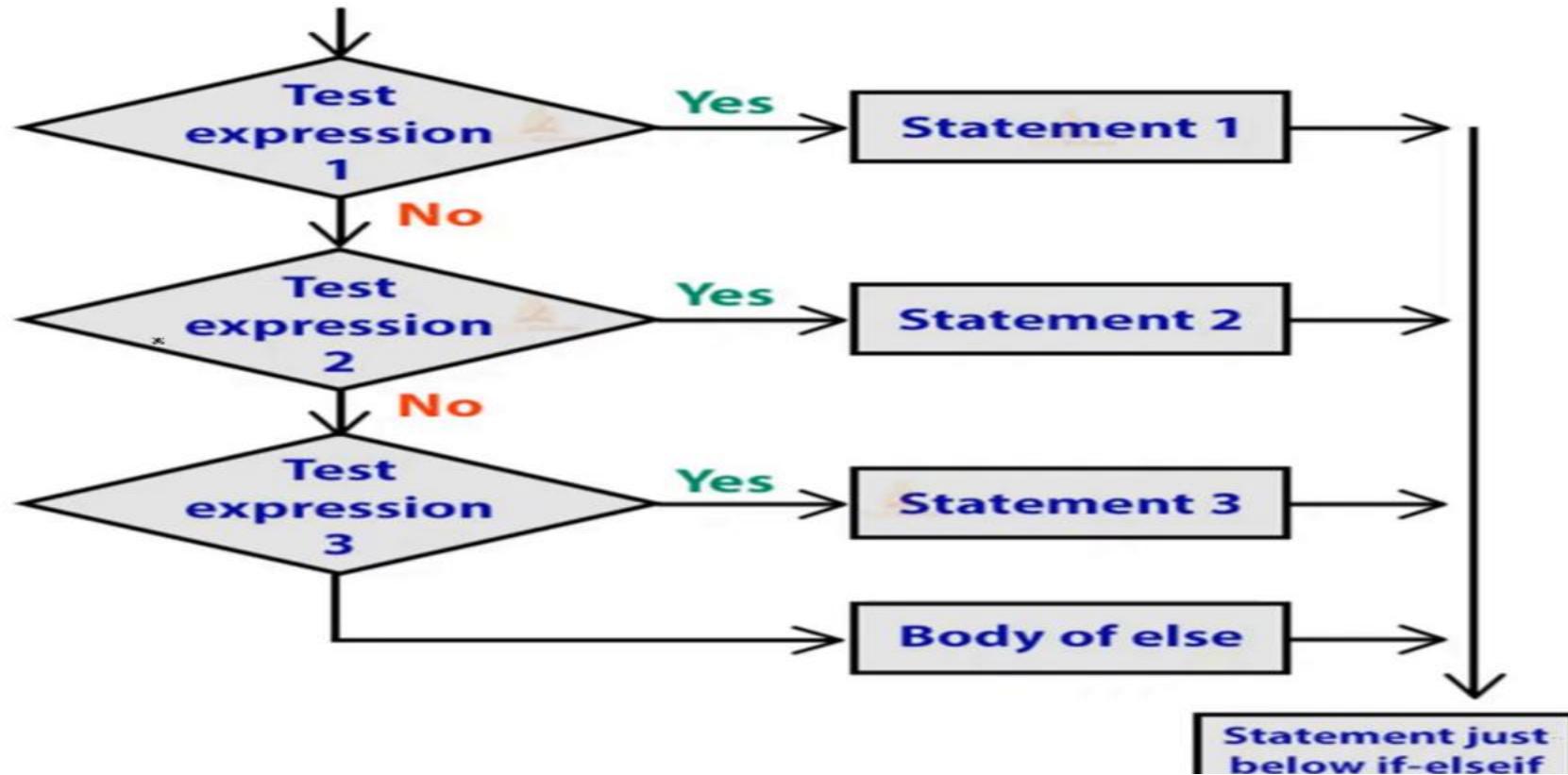
- Only one block is executed at one time.

elif in python:-

- The elif statement enables us to check multiple conditions and execute a specific block of statements depending upon True condition among them.

Python-Basics- Decision Making

Python if-elif ladder



Python-Basics- Decision Making

syntax:-

```
if test_expression1:  
    #block of statements  
elif test_expression2:  
    #block of statements  
elif test_expression3:  
    #block of statements  
else:  
    #block of statements
```

Nested if in Python:-

- We can have a **if-else** statements inside another if-else statement. This is called **nesting of if's**.
- Any number of these statements can be nested into one another.
- **Indentation** is the only way to figure out the level of nesting.
- **Confusing**, so avoid it.

Python-Basics- Decision Making

Syntax:-

*

```
if condition:  
    statement  
    statement  
    if condition:  
        statement  
        statement  
    else:  
        statement  
        statement  
    else:  
        statement  
#rest of code
```



Nested if-else

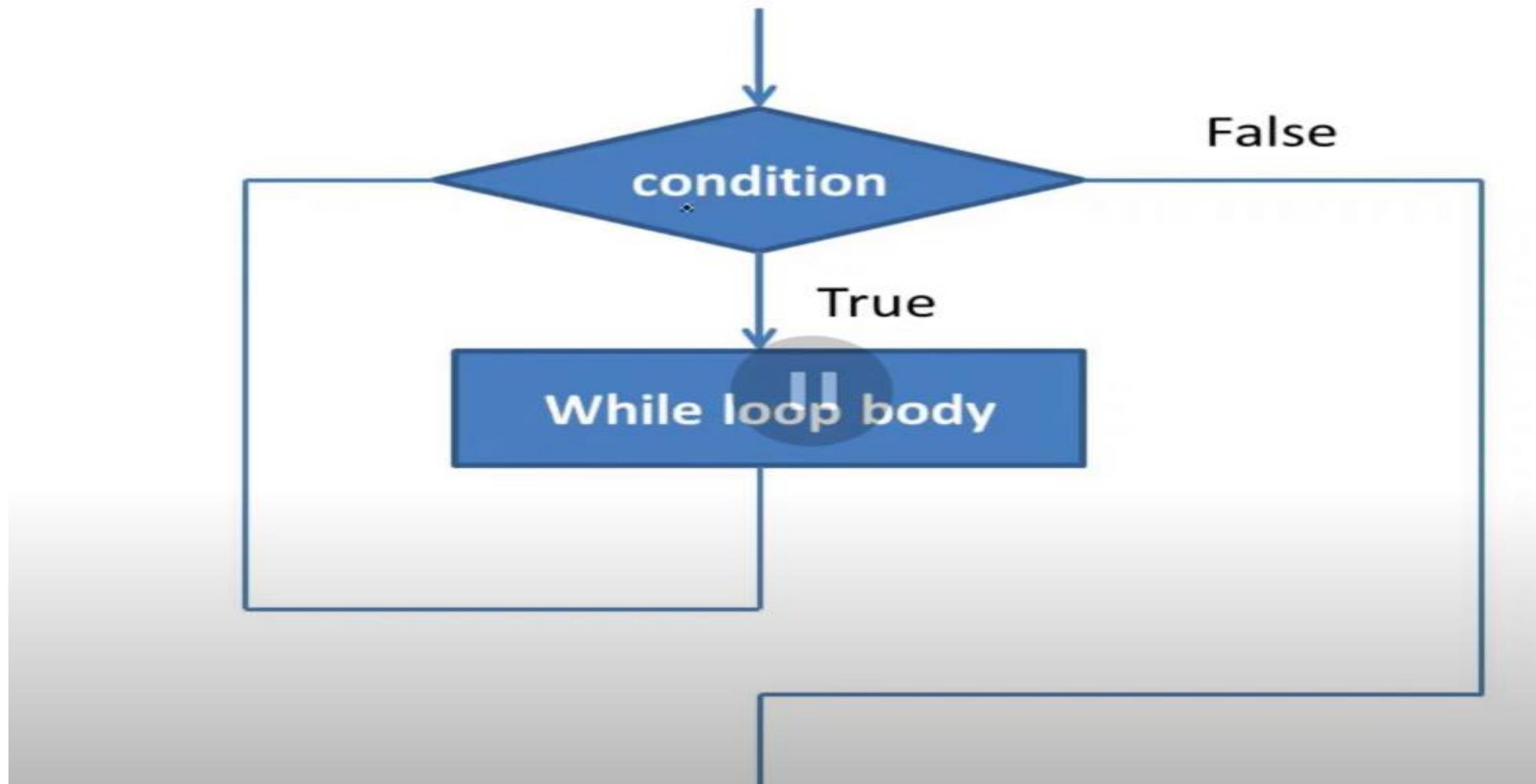
Python-Basics

- Loops are used in programming to repeat a specific block of code.
- In python,we have two types of loops:-
 - 1) While loop
 - 2) For loop

Python While loop:-

- syntax:-
while condition:
 block of code
- While loop is used to iterate over a block of code as long as condition is **True**.
- In the while loop, condition is checked first. If condition is **True**, body of while is executed.

Python-Basics



for loop:-

- for loop is used to iterate over a sequence (list,tuple,string) or other iterable objects.

syntax:-

```
for var in sequence:
```

```
    statement 1
```

```
    statement 2
```

```
    statement n
```

```
#Rest of the code
```

Python **break** :-

- In python,break statement used to alter the flow of program.
- When break occurs,control of program flows to statements immediately after body of loop.

syntax:-

```
for var in sequence:  
    #code inside for loop  
    if condition:  
        break  
    #code inside loop  
#rest of statements
```

```
while condition:  
    #code inside while loop  
    if condition:  
        break  
    #codes inside while loop  
#rest of statements
```

- Concatenation of strings.
- Multiplication of string
- Deletion of string.
- Iteration through string
- Checking character/substring present in string.

concatenation:-

- Joining two or more strings into a single one.
- It adds strings together.
- The '**+**' operator does this.
- Example:-

```
str1 = 'Python is'  
str2 = 'awesome'  
str3 = str1 + str2  
print(str3)
```

Output:-

◦ Python isawesome

Multiplication:-

- Repeat same string for multiple times.
- Using ‘*’
- Example:-

```
str1 = 'codeyug'  
str2 = str1*2  
print(str2)
```

Output:-
codeyugcodeyug

Deletion:-

- We cannot delete characters of string. But, deleting entire string is possible using ‘del’ keyword.
- Example:-

```
str1 = 'codeyug'  
del str1  
prints (str1)
```

- Iteration on string
- Finding length of string
- len() function
- String membership
- Finding number of vowels in entered string.

Finding length of string:-

- Length of string is nothing but number of characters in string.
- Two ways of finding length:-
 - 1) By manual programming
 - 2) By len() function.

Length by manual programming :-

```
str1 = input("Enter the string:")
counter=0
for char in str1:
    counter+=1
print('Length of string:',counter)
```



count():-

- Inbuilt method which returns a number of times a specified substring occurs in a string

Syntax:-

string.count(substring,start,end)

- Substring :- A string whose occurrences is to be calculated.
- start :- From where to start searching.(start index)
- end :- where to end searching.(end index)

important:-

- start,end are optional.
- Substring is required argument.
- Default start is index 0.
- Default end is end of string(index length of string)

Counting number of occurrences:-

```
str1 = input("Enter the string:")  
counter = str1.count('Co')  
print('Length of string:',counter)
```

2

counter

The diagram illustrates the string "Code" repeated twice. The first occurrence of "Code" is highlighted with orange rectangular boxes around each letter ("C", "o", "d", "e"). The second occurrence of "Code" is shown below it without any highlighting.

Counting number of occurrences:-

```
str1 = input("Enter the string:")  
counter = str1.count('Co',3,7)  
print('Length of string:',counter)
```

1

counter

c o d e c a d e

Python-Basics

```
1 world_cup_winners='''1975-WestIndies,1979-WestIndies,1983-India  
2 1987-Australia,1992-Pakistan,1996-SriLanka,1999-Australia,  
3 2003-Australia,2007-Australia,2011-India,2015-Australia,2019-Engla  
4 world_cup_india=world_cup_winners.count('India')  
5 print("India win time:",world_cup_india)
```

startswith():-

- This inbuilt method returns **True** if string starts with specified **prefix**. If not, returns **False**.

Syntax:-

`string.startswith(prefix,start,end)`

- **Substring** :- A string or tuple to be checked.
- **start** :- starts index from where searching starts.
- **end** :- End index of searching.

- Removing leading and trailing characters from given string.
- Three important inbuilt methods:
 - 1) `lstrip()`
 - 2) `strip()`
 - 3) `rstrip()`

- Removing leading and trailing characters from given string.
- Three important inbuilt methods:
 - 1) `lstrip()`
 - 2) `strip()`
 - 3) `rstrip()`

Istrip():-

- It removes any leading characters (space is the default leading characters to remove.)

Syntax:-

string.**Istrip(characters)**

- **characters :-** a set of characters to remove as leading characters . It is optional argument.
- All combinations of characters are removed from left until first mismatch.

strip():-

- It removes leading and trailing characters (space is the default characters to remove.)

Syntax:-

string.**strip(characters)**

- **characters** :- a set of characters to remove . It is optional argument.
- All combinations of characters are removed from left until first mismatch . Then,it remove from right until first mismatch.

- Python string comparison can be done using comparison operators (`==`,`!=`,`>`,`<`,`>=`,`<=`)
- The characters of both strings are compared one by one.
- When different characters are found,then their unicode values is compared.
- The character with lower unicode value is considered as lower.



replace():-

- The replace() method returns a copy of string where all occurrences of a sub-string is replaced with another substring

Syntax:-

```
string.replace(old_value,new_value,count)
```

- old_value** :- old substring you want to replace.
- new_value**:- new substring for old one.
- count**:- optional argument. Number of times you want to replace.

find():-

- The `find()` method returns index of first occurrence of substring. If not found, it returns `-1`.

Syntax:-

```
string.find(substring,start,end)
```

- `substring`:-The substring to search for.
- `start`:-where to start search . Default is 0.
- `end`:- Where to end the search . Default is end of string

rfind():-

- The rfind() method returns index of last occurrence of substring. If not found, it returns -1.

Syntax:-

string.**rfind**(substring,start,end)

split():-

- The `split()` method breaks up a string at specified separator and returns a list of strings.

Syntax:-

`string.split(separator,maxsplit)`

- separator**:- separator for splitting string. By default, whitespace is a separator
- maxsplit**:- how many splits to do.

List in python:-

- A list is a collection of values or items of different types.
- It is ordered collection.

Creating list:-

- The items in list are separated by comma(,) and enclosed within square brackets [].
ex. data=[20,13.4,'india',2+3j]

Note:- list and array are not same.

array:- stores data of similar type.

list:- stores data of different types.

How to implement array?:-

by using **array** module.

by using **numpy** package.

Accessing elements in list:-

Two ways:-

- 1) Indexing
- 2) Slicing

Indexing:- used to access individual(single) element of list.

slicing:- used to access group of elements of a list. We can access range of elements using slicing.

Index:-

- In python, Every element is represented by index.

```
list1 = ['yes',10,20.4,2+3j,[1,2,3]]
```

Negative indexing / Reverse Indexing



Positive indexing / Forward Indexing

Accessing Single Element:-

- To access single element of list, python uses indexing.

```
list1 = ['yes',10,20.4,2+3j,[1,2,3]]
```

-5	-4	-3	-2	-1
yes	10	20.4	2+3j	[1,2,3]
0	1	2	3	4

Syntax:- var_name[index]

list1[0] ---> yes

list1[1] ---> 10

list1[2] ---> 20.4

list1[-2] ---> 2+3j

list1[-1] ---> [1,2,3]

- Concatenation of lists
- Multiplication of lists
- Iteration through list
- Membership of list
- Deletion of list

finding maximum in list

- To find maximum in list , we use an inbuilt method of list i.e **max()**.

- **Syntax:-**

```
max(iterable,key,default)
```

iterable:- an iterable such as list.

Key (optional) :- basis for comparison.

Default:- default value is given iterable is empty.

finding minimum in list

- To find minimum in list , we use an inbuilt method of list i.e `min()`.
- **Syntax:-**
`min(iterable,key,default)`

iterable:- an iterable such as list.

Key (optional) :- basis for comparison.

Default:- default value is given iterable is empty.

Python list append():-

- The append() method adds a single item to list.
- It appends an element by modifying list.
- **Syntax:-**
`list.append(item)`
 - Item can be of any type.[✓]
- This method does not return any value rather modifies the list.Means,it returns ‘None’.

Python list extend():-

- The `append()` method extends list by appending all items from iterable. Iterable can be list,tuple,set etc
- **Syntax:-**
`list1.extend(iterable)`
- This method does not return any value rather modifies the list.Means,it returns 'None'.

Python list insert():-

- The `insert()` method insert an element to list at specified index.
- **Syntax:-**
`list.insert(index,element)`
 - `element` can be of any type.
- This method does not return any value rather modifies the list.Means,it returns 'None'.

Python-Basics

- **LIST**

- Remove()
- Pop()
- Clear()
- Del

- 1. **remove(element)**

Purpose: Removes the first occurrence of a specified element from the list.

Argument: Takes the element to be removed as an argument.

Effect: Modifies the original list in place.

Raises ValueError: If the element is not found in the list.

Example:

```
my_list = [10, 20, 30, 20, 40]
my_list.remove(20) # Removes the first occurrence of 20
print(my_list) # Output: [10, 30, 20, 40]
```

- **2. pop([index])**

Purpose: Removes and returns the element at the specified index. If no index is provided, it removes and returns the last element.

Argument: Optional index of the element to be removed.

Effect: Modifies the original list in place.

Raises IndexError: If the index is out of range.

Example:

```
my_list = [10, 20, 30, 40]
removed_element = my_list.pop(1) # Removes and returns the element at index 1 (20)
print(removed_element) # Output: 20
print(my_list) # Output: [10, 30, 40]
```

- **3. clear()**

Purpose: Removes all elements from the list, making it empty.

Argument: Takes no arguments.

Effect: Modifies the original list in place.

Example:

```
my_list = [10, 20, 30, 40]
my_list.clear()
print(my_list) # Output: []
```

Python-Basics

4.del

Purpose: Can be used to delete an element at a specific index, a slice of elements, or the entire list.

Argument: Takes the index, slice, or the list variable itself as an argument.

Effect: Modifies the original list in place or deletes the list variable entirely.

Raises IndexError: If the index is out of range.

Examples:

```
my_list = [10, 20, 30, 40]
del my_list[1] # Deletes the element at index 1 (20)
print(my_list) # Output: [10, 30, 40]
```

```
del my_list[1:3] # Deletes elements from index 1 to 2 (30, 40)
print(my_list) # Output: [10]
```

```
del my_list # Deletes the entire list variable
# print(my_list) # This would raise a NameError because my_list no longer exists
```

Python-Basics

Summary:

remove(): Removes the first occurrence of a specific element.

pop(): Removes and returns an element at a specific index (or the last element if no index is given).

clear(): Removes all elements, making the list empty.

del: Can be used to delete elements by index, slice, or the entire list variable.

What are Tuples?

- Tuples are ordered, **immutable** collections of elements.
- They are similar to lists, but unlike lists, tuples cannot be modified after creation.
- Tuples are defined using parentheses () and elements are separated by commas.
- Elements in a tuple can be of different data types.

Python-Basics

Creating Tuples

- Empty Tuple: `empty_tuple = ()`
- Tuple with Elements: `my_tuple = (1, "hello", 3.14)`
- Tuple with a Single Element: `single_tuple = (1,)` (Note the trailing comma)
- Using the `tuple()` constructor: `tuple_from_list = tuple([1, 2, 3])`

Accessing Elements

- Elements in a tuple are accessed using indexing, similar to lists.
- Indexing starts from 0 for the first element.
- Negative indexing can be used to access elements from the end of the tuple.

Immutability

- Once a tuple is created, its elements cannot be changed.
- This means you cannot add, remove, or modify elements in a tuple.
- Immutability provides data integrity and prevents accidental modifications.

Python-Basics

Operations on Tuples

- Concatenation: tuple1 + tuple2 creates a new tuple by combining the elements of tuple1 and tuple2.
- Repetition: tuple * n creates a new tuple by repeating the elements of tuple n times.
- Membership: element in tuple checks if element is present in tuple.
- Iteration: You can iterate through the elements of a tuple using a for loop.

When to Use Tuples

- When you need an ordered collection of elements that should not be modified.
- For representing data that is logically grouped together, such as coordinates or database records.
- As keys in dictionaries, since tuples are immutable and hashable.
- For returning multiple values from a function.

Advantages of Tuples

- **Data Integrity:** Immutability ensures that the data in a tuple remains unchanged.
- **Performance:** Tuples are generally faster than lists for accessing elements due to their immutability.
- **Hashability:** Tuples can be used as keys in dictionaries, which is not possible with lists.

Memory Allocation: List Vs Tuple

Lists: Lists are mutable, meaning their size and contents can be changed after creation.

To accommodate potential changes, Python allocates extra memory to lists beyond the space required to store the initial elements. This extra memory allows for efficient insertion and deletion of elements without frequent reallocations.

Tuples: Tuples are immutable, meaning their size and contents are fixed once they are created. Since tuples cannot be modified, Python allocates only the minimum memory block required to store their data. This results in tuples being more memory-efficient compared to lists.

What is a dictionary?

- An unordered collection of key-value pairs.
- Keys must be unique and immutable (e.g., strings, numbers, tuples).
- Values can be of any data type.
- Denoted by curly braces { }.

Why use dictionaries?

- Efficient way to store and retrieve data based on keys.
- Useful for representing real-world objects with attributes.
- Flexible and adaptable data structure.

Python-Basics

Accessing Values

- Using keys: `my_dict["name"]` (returns "Alice")
- Using the `get()` method: `my_dict.get("age")` (returns 30)
- `get()` method avoids `KeyError` if the key is not found.
- Checking for key existence: "name" in `my_dict` (returns True)

Modifying Dictionaries

- Adding new key-value pairs: `my_dict["occupation"] = "Engineer"`
- Updating existing values: `my_dict["age"] = 31`
- Removing key-value pairs:
 - `del my_dict["city"]`
 - `my_dict.pop("age")` (returns the removed value)

Dictionary Methods

- `keys()`: Returns a list of all keys.
- `values()`: Returns a list of all values.
- `items()`: Returns a list of key-value pairs as tuples.
- `clear()`: Removes all key-value pairs.
- `copy()`: Creates a shallow copy of the dictionary.

What is set?

- A set is an unordered collection of items of different types enclosed in curly braces.
- Ex.

```
s = {101,'shantanu','pass',98.2}
```

Creating set:-

Two ways:-

1) using curly braces.

syntax:- setname = {ele1,ele2,ele3}

2) using set() function.

syntax:- setname = set(iterable)

Important about set:-

- 1. No duplicates are allowed.

Ex.

```
s={10,20,10,10,20,30,20}
```

```
print(s)
```

o/p:- {10,20,30}

- 2. Elements are not in particular order.

Ex.

```
s={10,20,10,10,20,30,20}
```

```
print(s)
```

o/p:- {10,20,30} / {20,10,30} / {30,10,20} /anything

- 3. Indexing and slicing cannot be applied on set.

Ex.

```
s={10,20,10,10,20,30,20}  
print(s[1])
```

o/p:- TypeError: 'set' object is not subscriptable

➤ 4. Set is mutable.

Ex.

```
s={10,20,10,10,20,30,20}  
s.add(100)
```

o/p:- 100 will be added at any position.

- **5. Elements in set must be immutable.**

Ex.

```
s={10,20,10,10,[10,20,30]}\nprint(s)
```

o/p:- Type-Error: unhashable type: 'list'

Creating Empty set:-

➤ Syntax:-

```
set_name =set( )
```

Set with single item:-

➤ Syntax:-

```
set_name = {100}
```

Revision

Set in python

- Items are unique
- Set is mutable
- Items are immutable.
- Items are unordered
- No indexing and slicing

Python-Basics

Real-World Use Cases for Sets:

- **1. Removing Duplicates:** Sets are excellent for removing duplicates from a list or any other iterable. Simply convert the iterable to a set, and the duplicates will be automatically eliminated.
- **2. Membership Testing:** Checking if an element exists in a large collection is much faster with sets compared to lists. This is useful for tasks like validating user input or filtering data.
- **3. Mathematical Operations:** Sets provide efficient implementations of set operations like union, intersection, difference, and symmetric difference. These are useful for tasks like finding common elements, identifying unique elements, or comparing data sets.
- **4. Data Deduplication:** In databases or data analysis, sets can be used to identify and remove duplicate records.

01 **add() method**

- Add one element to set

02 **update() method**

- Add multiple elements in set

03 **remove() method**

- Remove one element



04 **discard() method**

- Remove one element

05 **pop() method**

- Remove any element randomly

06 **clear() method**

- Remove all elements from set



Adding item to set

- Add single item to set at random position.
- Syntax:

`set_name.add(item)`



Adding items to set

- Add multiple items to set at random positions.
- Syntax:

```
set_name.update([item1,item2])
```

Removing single item

- Remove single item.
- Syntax:

```
set_name.discard(item)  
set_name.remove(item)
```



Removing random item

- Remove single item randomly.

- Syntax:

```
set_name.pop()
```

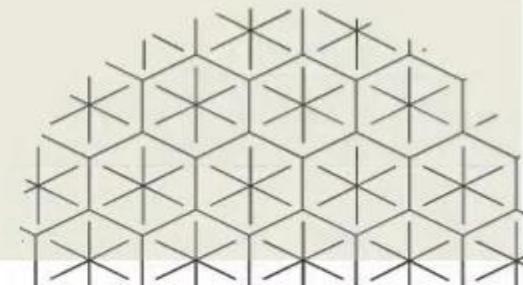
What is union of sets?

- Union of two given sets is **the smallest set which contains all the elements of both the sets**
- Ex.

$$A = \{2, 4, 5, 6\}$$

$$B = \{4, 6, 7, 8\}$$

output:- {2, 4, 5, 6, 7, 8}



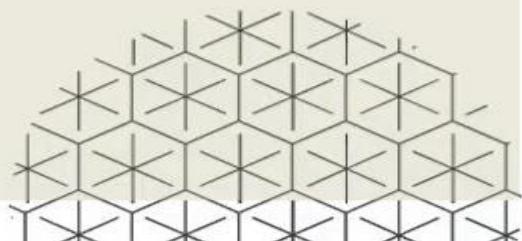
What is Intersection of sets?

- Intersection of two given sets is the largest set which contains all the elements that are common to both the sets.
- Ex.

$$A = \{2, 4, 5, 6\}$$

$$B = \{4, 6, 7, 8\}$$

output:- {4, 6}



What is Difference of sets?



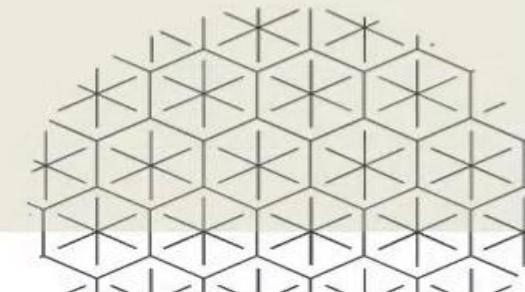
- The difference between the two sets in Python is **equal to the difference between the number of elements in two sets.**
- Ex.

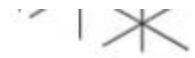
$$A = \{2, 4, 5, 6\}$$

$$B = \{4, 6, 7, 8\}$$

$$A-B : \{2,5\}$$

$$B-A : \{7,8\}$$





Frozen-set in python

What isFrozenSet?

- Frozen-set is an immutable version of python set.
- While elements of a set can be modified at any time, elements of the frozen set remain the same after creation.

How to create Frozenset?

- By using frozenset() inbuilt function.
- Syntax: frozenset(iterable)

Python-Basics

Q1

1. Which of the following is the correct way to access the third element of a list named my_list?
 - a) my_list[2]
 - b) my_list[3]
 - c) my_list[-1]
 - d) my_list[-3]

Python-Basics

Q1

1. Which of the following is the correct way to access the third element of a list named my_list?
 - a) my_list[2]
 - b) my_list[3]
 - c) my_list[-1]
 - d) my_list[-3]

Python-Basics

Q2

. my_list = [1, 2, 3, 4, 5]
 print(my_list[1:3])

a) [1, 2]

b) b) [2, 3]

c) c) [2, 3, 4]

d) d) [1, 2, 3]

Python-Basics

Q2

. my_list = [1, 2, 3, 4, 5]
 print(my_list[1:3])

a) [1, 2]

b) b) [2, 3]

c) c) [2, 3, 4]

d) d) [1, 2, 3]

Python-Basics

Q3

. 3. Which list method is used to add an element to the end of a list?

- a) insert()
- b) b) append()
- c) c) extend()
- d) d) add()

Python-Basics

Q3

. 3. Which list method is used to add an element to the end of a list?

- a) insert()
- b) b) append()
- c) c) extend()
- d) d) add()

Python-Basics

Q4

- . 4. What is the output of the following code snippet?

```
my_list = [1, 2, 3]
my_list.extend([4, 5])
print(my_list)
```

- a) [1, 2, 3, 4, 5]
- b) [1, 2, 3, [4, 5]]
- c) [1, 2, 3, 4]
- d) [1, 2, 3, 5]

Python-Basics

Q4

- . 4. What is the output of the following code snippet?

```
my_list = [1, 2, 3]
my_list.extend([4, 5])
print(my_list)
```

- a) [1, 2, 3, 4, 5]
- b) [1, 2, 3, [4, 5]]
- c) [1, 2, 3, 4]
- d) [1, 2, 3, 5]

Python-Basics

Q5

- . 5. Which list method is used to remove an element by its value?
a) pop() b) remove() c) delete() d) clear()

Python-Basics

Q5

- . 5. Which list method is used to remove an element by its value?
a) pop() b) **remove()** c) delete() d) clear()

Python-Basics

Q6

- . 6. Which list method is used to insert an element at a specific index?
 - a) append() b) extend() c) insert() d) add()

Python-Basics

Q6

- . 6. Which list method is used to insert an element at a specific index?
 - a) append() b) extend() c) **insert()** d) add()

Python-Basics

Q7

- . 7. What is the output of the following code snippet?

```
my_list = [1, 2, 3, 4, 5]
print(my_list[::-1])
```

- a) [1, 2, 3, 4, 5]
- b) b) [5, 4, 3, 2, 1]
- c) c) [1, 3, 5]
- d) d) [2, 4]

Python-Basics

Q7

- . 7. What is the output of the following code snippet?

```
my_list = [1, 2, 3, 4, 5]
print(my_list[::-1])
```

- a) [1, 2, 3, 4, 5]
- b) b) [5, 4, 3, 2, 1]
- c) c) [1, 3, 5]
- d) d) [2, 4]

Python-Basics

Q8

- . 8. What is the output of the following code snippet?

```
my_list = [1, 2, 3, 4, 5]
new_list = my_list
new_list[0] = 10
print(my_list)
```

- a) [10, 2, 3, 4, 5]
- b) [1, 2, 3, 4, 5]
- c) [1, 10, 3, 4, 5]
- d) [10, 10, 3, 4, 5]

Python-Basics

Q8

- . 8. What is the output of the following code snippet?

```
my_list = [1, 2, 3, 4, 5]
new_list = my_list
new_list[0] = 10
print(my_list)
```

- a) [10, 2, 3, 4, 5]
- b) [1, 2, 3, 4, 5]
- c) [1, 10, 3, 4, 5]
- d) [10, 10, 3, 4, 5]

Python-Basics

Q9

- . 9. What is the output of the following code snippet?

```
my_list = [1, 2, [3, 4]]  
print(my_list[2][0])
```

- a) 1
- b) 2
- c) 3
- d) 4

Python-Basics

Q9

- . 9. What is the output of the following code snippet?

```
my_list = [1, 2, [3, 4]]  
print(my_list[2][0])
```

- a) 1
- b) 2
- c) 3
- d) 4

Python-Basics

Q10

. 10. What is the result of the expression [1, 2, 3] * 3?

a) [1, 2, 3, 1, 2, 3, 1, 2, 3]

b) [3, 6, 9]

c) [[1, 2, 3], [1, 2, 3], [1, 2, 3]]

d) Error

Python-Basics

Q10

. 10. What is the result of the expression [1, 2, 3] * 3?

- a) [1, 2, 3, 1, 2, 3, 1, 2, 3]
- b) [3, 6, 9]
- c) [[1, 2, 3], [1, 2, 3], [1, 2, 3]]
- d) Error

Q11

11. Which of the following is the correct way to create an empty dictionary?

a) {}

b) []

c) ()

d) dict()

Python-Basics

Q11

11. Which of the following is the correct way to create an empty dictionary?

a) {}

b) []

c) ()

d) dict()

Python-Basics

Q12

12. What is the output of the following code snippet?

```
my_dict = {'a': 1, 'b': 2, 'c': 3}  
print(my_dict['b'])
```

- a) 1
- b) 2
- c) 3
- d) KeyError

Python-Basics

Q12

12. What is the output of the following code snippet?

```
my_dict = {'a': 1, 'b': 2, 'c': 3}  
print(my_dict['b'])
```

- a) 1
- b) 2
- c) 3
- d) KeyError

Python-Basics

Q13

1 Which dictionary method is used to add a new key-value pair?

- a) append()
- b) insert()
- c) update()
- d) There is no specific method; you can directly assign a value to a new key.

Python-Basics

Q13

1 Which dictionary method is used to add a new key-value pair?

- a) append()
- b) insert()
- c) update()
- d) There is no specific method; you can directly assign a value to a new key.

Q14

14. Which dictionary method is used to add a new key-value pair?

- a) append()
- b) insert()
- c) update()
- d) There is no specific method; you can directly assign a value to a new key.

Q14

14 Which dictionary method is used to add a new key-value pair?

- a) append()
- b) insert()
- c) update()
- d) There is no specific method; you can directly assign a value to a new key.

Python-Basics

Q15

15 Which of the following is NOT a valid dictionary key?

- a) 10
- b) "hello"
- c) [1, 2, 3]
- d) (1, 2).

Python-Basics

Q15

15 Which of the following is NOT a valid dictionary key?

- a) 10
- b) "hello"
- c) [1, 2, 3]
- d) (1, 2).

Python-Basics

Q16

16 How can you check if a key exists in a dictionary?

- a) Use the `in` keyword.
- b) Use the `has_key()` method.
- c) Use the `find()` method.
- d) Use the `index()` method.

Python-Basics

Q16

16 How can you check if a key exists in a dictionary?

- a) Use the `in` keyword.
- b) Use the `has_key()` method.
- c) Use the `find()` method.
- d) Use the `index()` method.

Q17

Which of the following is the correct way to create an empty tuple?

- a) ()
- b) []
- c) {}
- d) tuple()

Q17

Which of the following is the correct way to create an empty tuple?

- a) ()
- b) []
- c) {}
- d) tuple()

Python-Basics

Q18

Which of the following operations is NOT allowed on a tuple?

- a) Concatenation using the + operator.
- b) Repetition using the * operator.
- c) Slicing using the : operator.
- d) Modifying an element using indexing (e.g., my_tuple[0] = 10).

Q18

Which of the following operations is NOT allowed on a tuple?

- a) Concatenation using the + operator.
- b) Repetition using the * operator.
- c) Slicing using the : operator.
- d) Modifying an element using indexing (e.g., `my_tuple[0] = 10`).

Python-Basics

Q19

19 Why are tuples often preferred over lists in certain situations?

- a) They are more memory-efficient.
- b) They can be used as dictionary keys.
- c) They provide data integrity when immutability is desired.
- d) All of the above.

Q19

19 Why are tuples often preferred over lists in certain situations?

- a) They are more memory-efficient.
- b) They can be used as dictionary keys.
- c) They provide data integrity when immutability is desired.
- d) All of the above.

Python-Basics

Q20

Which of the following statements accurately describes the key difference between tuples and lists?

- a) Tuples are ordered collections, while lists are unordered.
- b) Tuples are mutable, while lists are immutable.
- c) Tuples are immutable, while lists are mutable.
- d) Tuples can only contain homogeneous elements, while lists can contain heterogeneous elements.

Python-Basics

Q20

Which of the following statements accurately describes the key difference between tuples and lists?

- a) Tuples are ordered collections, while lists are unordered.
- b) Tuples are mutable, while lists are immutable.
- c) Tuples are immutable, while lists are mutable.
- d) Tuples can only contain homogeneous elements, while lists can contain heterogeneous elements.