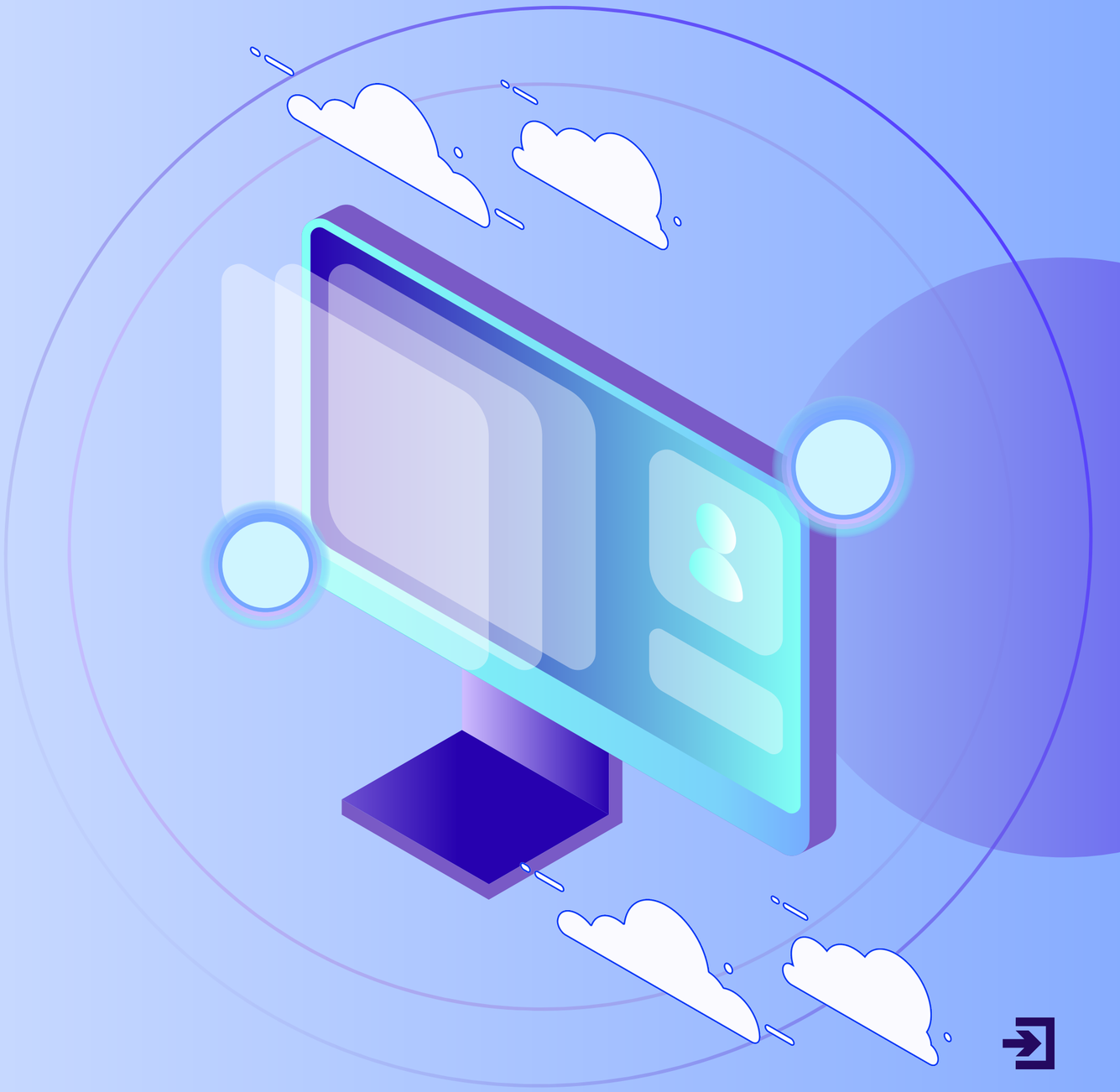
skill shikshya

PYTHON DJANGO

https://skillshikshya.com



IN PYTHON

TABLE OF CONTENTS

TABLE OF CONTENTS

INTRODUCTION TO PYTHON

WHAT IS PYTHON

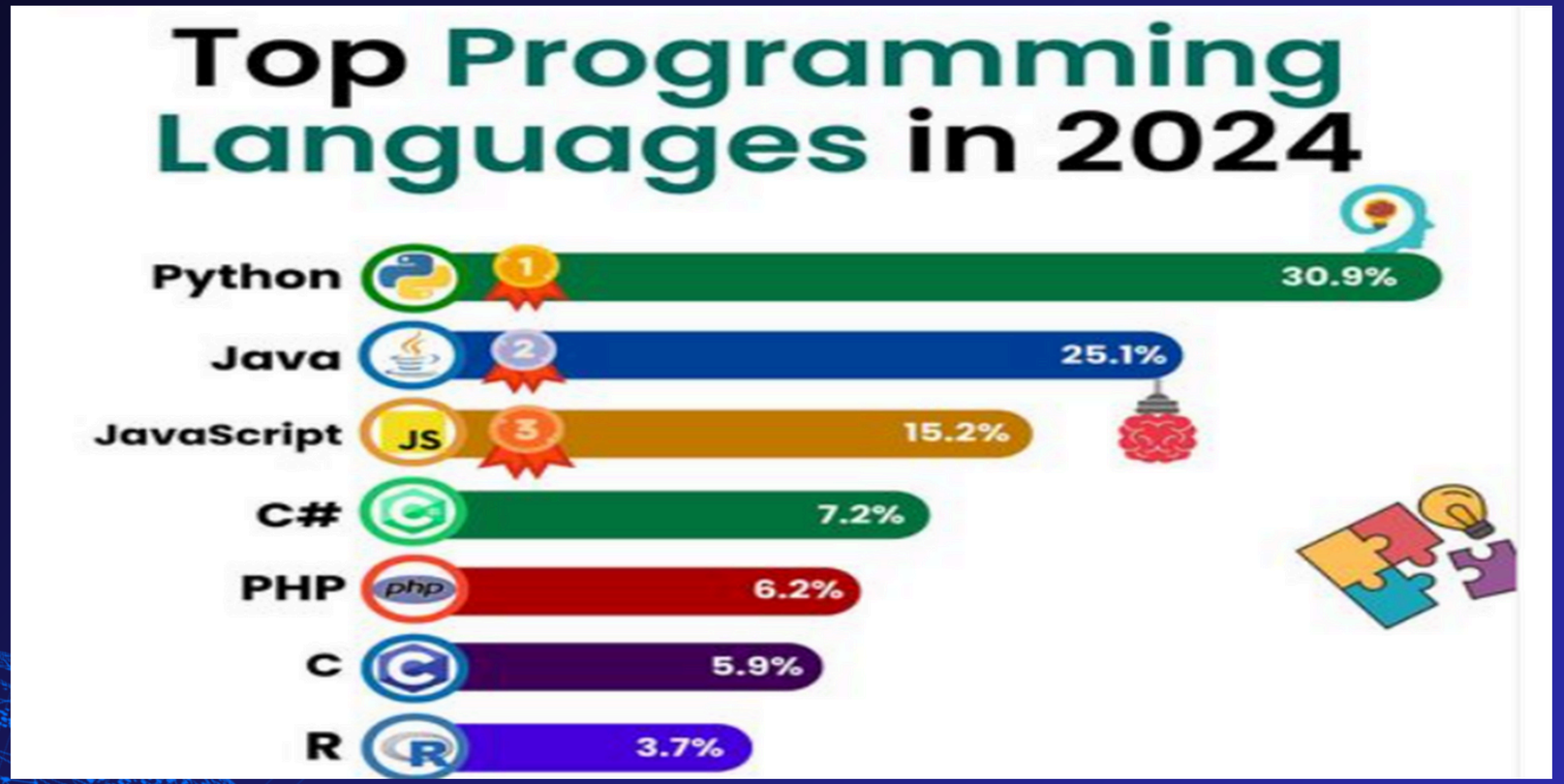
->Python is a high-level, interpreted programming language known for its simplicity and readability.

-->It was created by Guido van Rossum, and released in 1991.

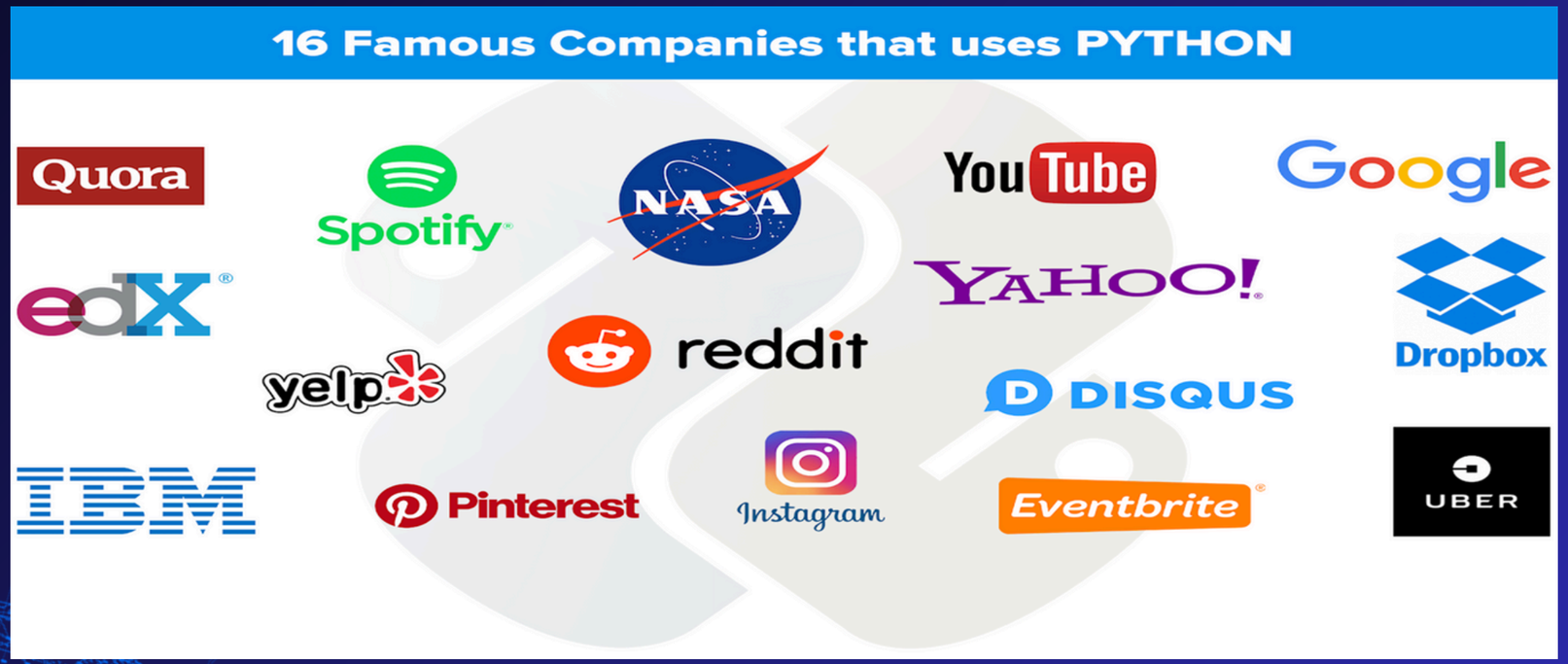
-->Python was named after the British comedy group Monty Python, not the snake. Guido van Rossum, Python's creator, is a fan of the group's comedy show. He chose the name because it was short, unique, and memorable.

SCOPE OF PYTHON

TOP PROGRAMMING LANGUAGE IN 2024

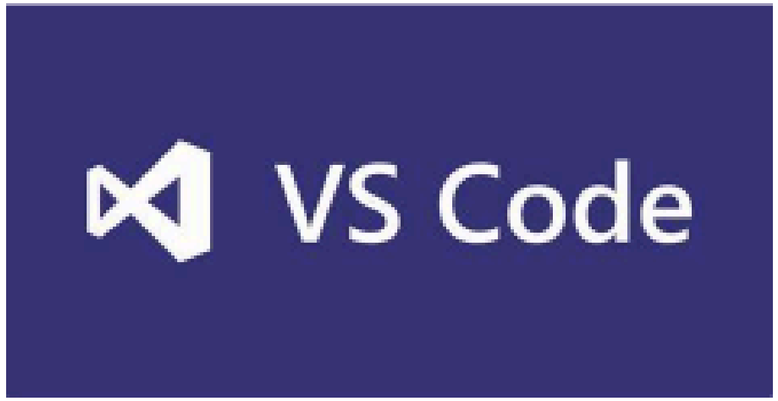


WHO USE PYTHON TODAY



Installing Python IDE do with

- **To install a Python** :Visit the website of the IDE you want to download.
- Click the download link.
- Once the download is complete, run the exe for install.
- Change the installation path if required.
- Create a desktop shortcut if you want and click on “Next”.



VARIABLES

varirable is a name which store value

a=5
name=”ram”

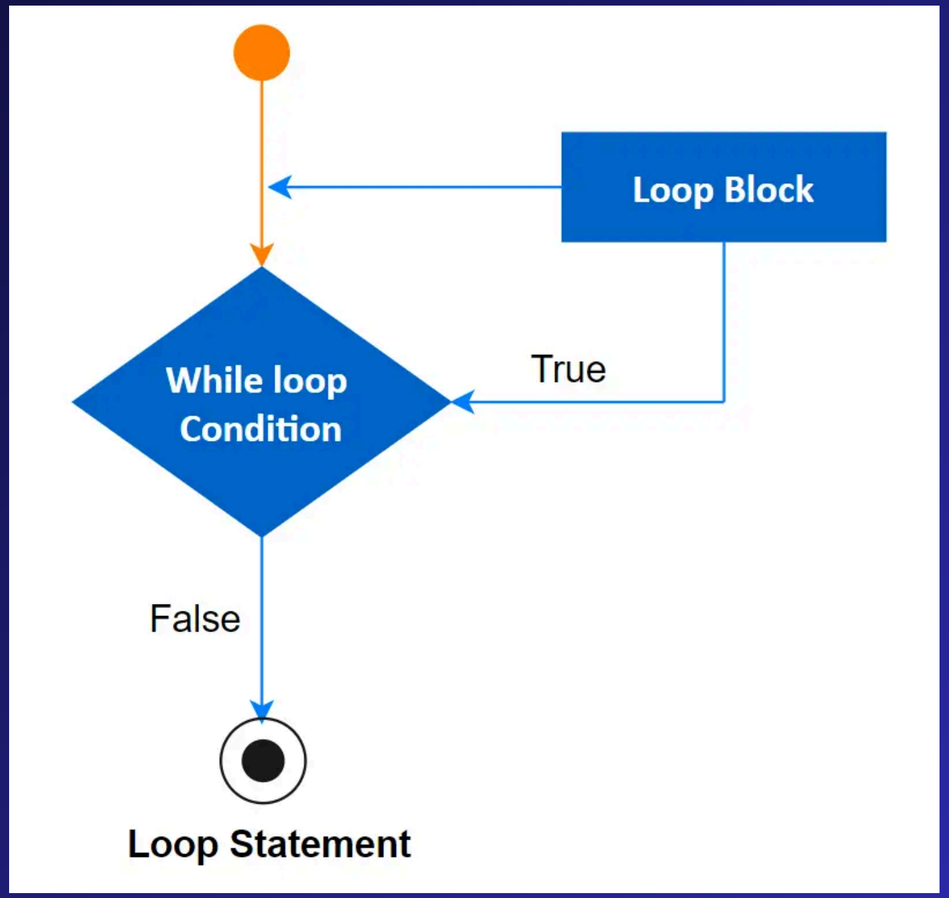
WHILE LOOP

In Python, a while loop is a control flow statement that allows you to execute a block of code repeatedly while a given condition is true.

The condition is evaluated at the start of each loop iteration, and if it is true, the loop body is executed

The loop is terminated if the condition is false.

WHILE LOOP



SYNTAX OF WHILE LOOP

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

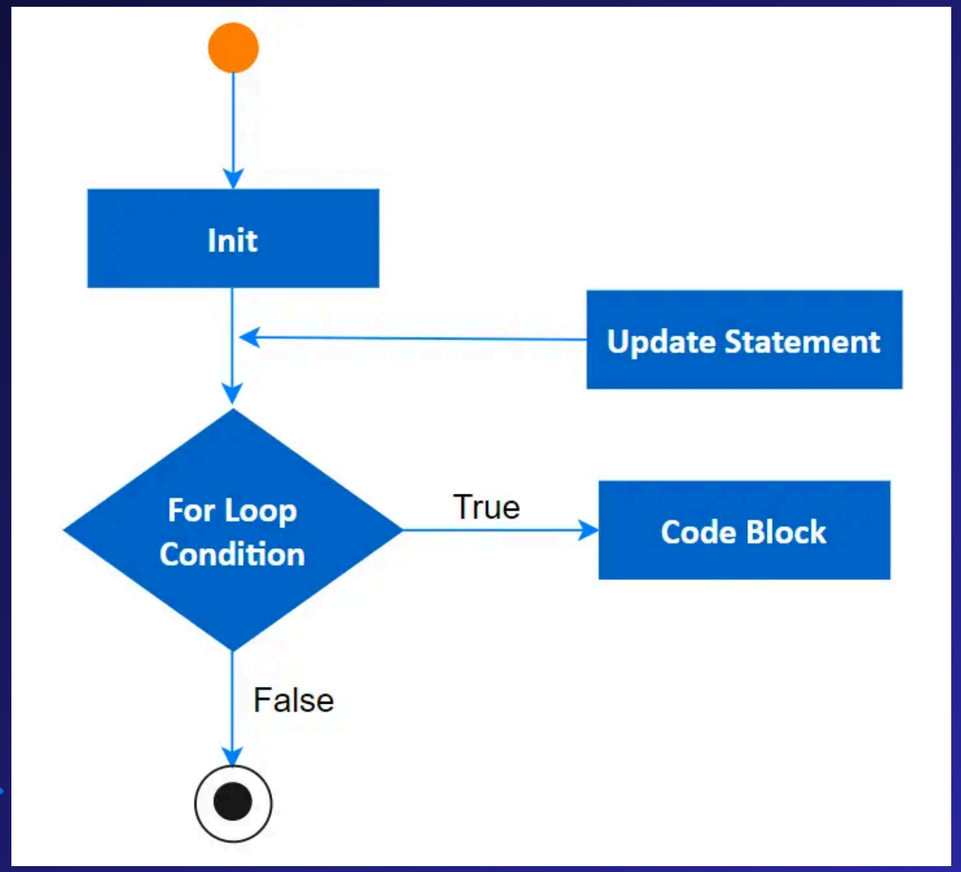
Example

```
count = 0
while (count < 10):
    print ('The count is:', count)
    count = count + 1
    print ("Good bye!")
```

FOR LOOP

- A for loop is a control flow statement in Python that allows you to iterate over a sequence of elements such as a list, tuple, or string.
    - On each iteration, the loop variable in Python will take on the value of the next item in the sequence.

FOR LOOP


Init
Update Statement
For Loop Condition
True
Code Block
False

SYNTAX OF FOR LOOP

```
for iterating_var in sequence:
    statements(s)
```

Example

```
for letter in 'Python': # First Example
    print ('Current Letter :', letter)

fruits = ['guava', 'apple', 'mango']
for fruit in fruits: # Second Example
    print ('Current fruit :', fruit)
print ("Good bye!")
```

FOR LOOP

Output

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : h
Current Letter : o
Current Letter : n
Current fruit : guava
Current fruit : apple
Current fruit : mango
Good bye!
```

NESTED LOOP

- In Python, a nested loop is a loop that is contained within another loop.
    - This indicates that the inner loop is executed within each outer loop iteration.
        - Nested loops are handy for iterating over a two-dimensional array or processing hierarchical data.

SYNTAX OF NESTED LOOP

```
for iterating_var in sequence:
    for iterating_var in sequence:
        statements(s)
```

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

Example

```
i = 2
while(i < 100):
   j = 2
   while(j <= (i/j)):
      if not(i%j): break
      j = j + 1
   if (j > i/j) : print (i, " is prime")
   i = i + 1
print ("Good bye!")
```

LOOP CONTROL

Break

continue

pass

BREAK

```
for i in range(10):
  if i == 5:
    break
  print(i)
```

The break statement is used to quickly stop the loop regardless of whether the loop condition is still true. This signifies that the loop will be terminated and the next statement outside of the loop will be executed.

Output

```
0
1
2
3
4
```

CONTINUE

```
for i in range(10):
  if i % 2 == 0:
    continue
  print(i)
```

The continue statement is used to skip the current loop iteration and go to the next iteration. This implies that the code from the current iteration will not be performed, and the loop will continue to iterate until the loop condition is satisfied.

Output

```
1
3
5
7
9
```

PASS

```
for i in range(10):
    # Do nothing
    pass
```

The pass statement is a null statement that has no effect. It can be utilized as a placeholder in control statements or empty loops.

Output

```
1
3
5
7
9
```

DATA STRUCTURE IN PYTHON

WHAT IS DATA STRCTURE

Data structures are specialized formats or ways of organizing and storing data in a computer's memory to facilitate efficient retrieval and manipulation of information

Data structures are specialized formats or ways of organizing and storing data in a computer's memory to facilitate efficient retrieval and manipulation of information

DATA STRUCTURE

- LIST
- TUPLE
- DICTIONARY
- SET
- LINKED LIST
- GRAPH
- STRING
- Queue
- Heap

LIST

List is a collection which is ordered and changeable.
Allows duplicate members

List is a collection which is ordered and changeable. Allows duplicate members

Lists are created using square brackets:

List items are ordered, changeable, and allow duplicate values.

List items are indexed, the first item has index [0], the second item has index [1] etc.

CREATE LIST

we can create list using 2 way.

- using list constructor list_1 = list(("ram",”shyam”,”hari”,”gopal”))
- using square bracket [ ] list_2 = [”ram", "gopal”, "hari"]

ACCESS LIST

list_1 = [1,2,3,4,5,6,9,23,34]

list_1[start:end:step]

DICTIONARY

A Python dictionary is a data structure that stores the value in key:value pairs.

A dictionary is a collection which is ordered*, changeable and do not allow duplicates.

Dictionaries are written with curly brackets, and have keys and values:

```
Dict = {’name: ‘skill shikshya', ‘location’: 'bagmati'}
print(Dict)
```

You can access the items of a dictionary by referring to its key name, inside square brackets:

DICTIONARY METHOD

dict.keys():- this return all keys

dict.values():-this return all values.

dict.items():-this return key pair data

dict.pop('key'):-this remove item

new_dict = dict.copy():-this copy to new dict

WHAT IS LIST COMPREHENSION

A Python list comprehension consists of brackets containing the expression, which is executed for each element along with the for loop to iterate over each element in the Python list.

```
numbers = [12, 13, 14,]
doubled = [x *2  for x in numbers]
print(doubled)
```

ENUMERATE() IN PYTHON

Often, when dealing with iterators, we also need to keep a count of iterations. Python eases the programmers' task by providing a built-in function enumerate() for this task. The enumerate () function adds a counter to an iterable and returns it in the form of an enumerating object. This enumerated object can then be used directly for loops or converted into a list of tuples using the list() function.

Syntax: enumerate(iterable, start=0)

```
l1 = ["eat", "sleep", "repeat"]
s1 = "geek"

# creating enumerate objects
obj1 = enumerate(l1)
obj2 = enumerate(s1)

print ("Return type:", type(obj1))
print (list(enumerate(l1)))

# changing start index to 2 from 0
print (list(enumerate(s1, 2)))
```

Output:

```
Return type: <class 'enumerate'>
[(0, 'eat'), (1, 'sleep'), (2, 'repeat')]
[(2, 'g'), (3, 'e'), (4, 'e'), (5, 'k')]
```

ENUMERATE OBJECT IN LOOPS

Often, when dealing with iterators, we also need to keep a count of iterations. Python eases the programmers' task by providing a built-in function enumerate() for this task. The enumerate () function adds a counter to an iterable and returns it in the form of an enumerating object. This enumerated object can then be used directly for loops or converted into a list of tuples using the list() function.

Syntax: enumerate(iterable, start=0)

```python
l1 = ["eat", "sleep", "repeat"]

# printing the tuples in object directly
for ele in enumerate(l1):
    print (ele)

# changing index and printing separately
for count, ele in enumerate(l1, 100):
    print (count, ele)

# getting desired output from tuple
for count, ele in enumerate(l1):
    print(count)
    print(ele)
```

ZIP IN PYTHON

Python zip() method takes iterable containers and returns a single iterator object, having mapped values from all the containers.

```
name = [ "Manjeet", "Nikhil", "Shambhavi", "Astha" ]
roll_no = [ 4, 1, 3, 2 ]

# using zip() to map values
mapped = zip(name, roll_no)

print(set(mapped))
```

{('Nikhil', 1), ('Shambhavi', 3), ('Manjeet', 4), ('Astha', 2)}

SET IN PYTHON

A Set in Python programming is an unordered collection data type that is iterable, mutable and has no duplicate elements.

Set are represented by { } (values enclosed in curly braces)

The major advantage of using a set, as opposed to a list, is that it has a highly optimized method for checking whether a specific element is contained in the set. This is based on a data structure known as a hash table. Since sets are unordered, we cannot access items using indexes as we do in lists.

```
# typecasting list to set
myset = set(["a", "b", "c"])
print(myset)
# Adding element to the set
myset.add("d")
print(myset)
```

UNION OPERATION

two sets can be merged using union() function or | operator. Both Hash Table values are accessed and traversed with merge operation perform on them to combine the elements, at the same time duplicates are removed

```
# Python Program to
# demonstrate union of
# two sets

people = {"Jay", "Idrish", "Archil"}
vampires = {"Karan", "Arjun"}
dracula = {"Deepanshu", "Raju"}

# Union using union()
# function
population = people.union(vampires)

print("Union using union() function")
print(population)

# Union using "|"
# operator
population = people|dracula

print("\nUnion using '|' operator")
print(population)
```

INTERSECTION OPERATION IN SET

This can be done through intersection() or & operator. Common Elements are selected. They are similar to iteration over the Hash lists and combining the same values on both the Table.

```
# Python program to
# demonstrate intersection
# of two sets

set1 = set()
set2 = set()

for i in range(5):
    set1.add(i)

for i in range(3,9):
    set2.add(i)

# Intersection using
# intersection() function
set3 = set1.intersection(set2)

print("Intersection using intersection() function")
print(set3)

# Intersection using
# "&" operator
set3 = set1 & set2

print("\nIntersection using '&' operator")
print(set3)
```

TUPLE IN PYTHON

A tuple is a collection similar to a Python List The primary difference is that we cannot modify a tuple once it is created.

We create a tuple by placing items inside parentheses ()

Tuple Characteristics

Tuples are:

Ordered - They maintain the order of elements.

Immutable - They cannot be changed after creation.

Allow duplicates - They can contain duplicate values.

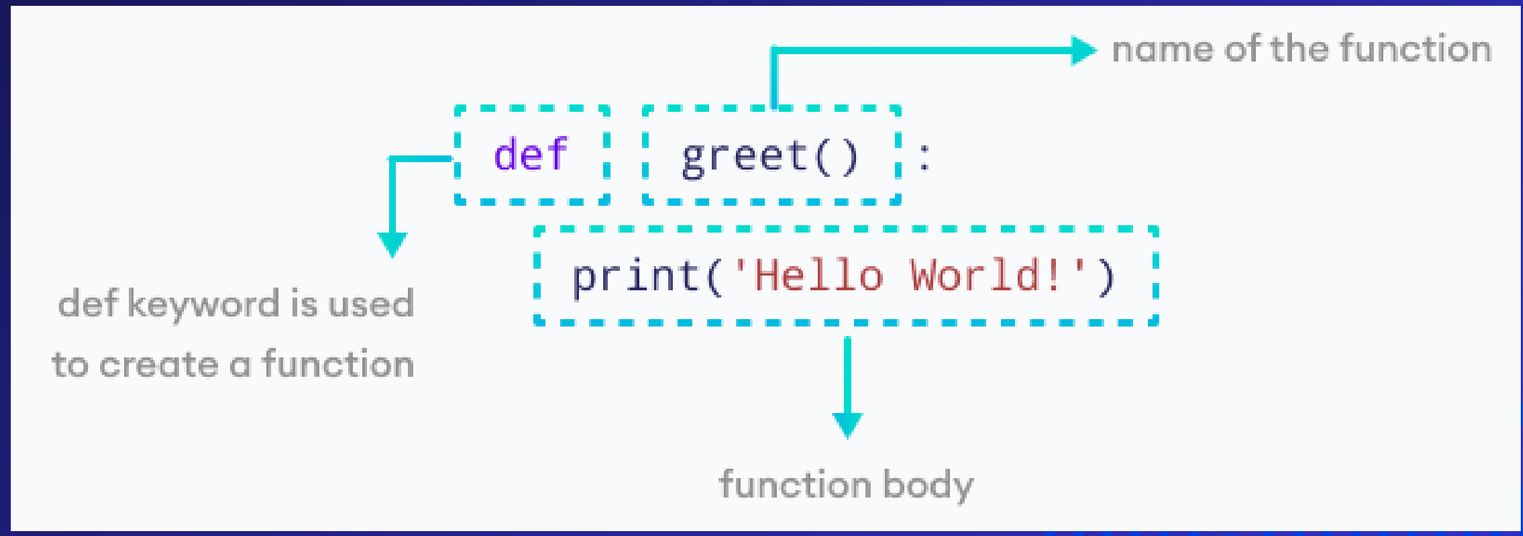
```
languages = ('Python', 'Swift', 'C++

# access the first item
print(languages[0])   # Python

# access the third item
print(languages[2])   # C++
```

FUNCTION IN PYTHON

a function is **a block of program statements which can be used repetitively in a program**. It saves the time of a developer. In Python concept of function is same as in other languages.

we can create python using def keyword.

syntax:
def function_name():
print(“**”)



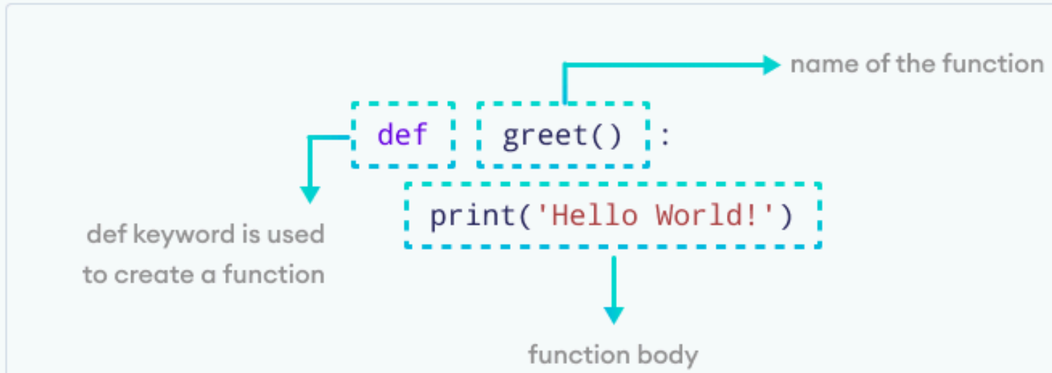
CREATE FUNCTION PYTHON

Create a Function

Let's create our first function.

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

Here are the different parts of the program:



LAMBDA FUNCTION

A lambda function is a small, anonymous function in Python that is defined using the lambda keyword

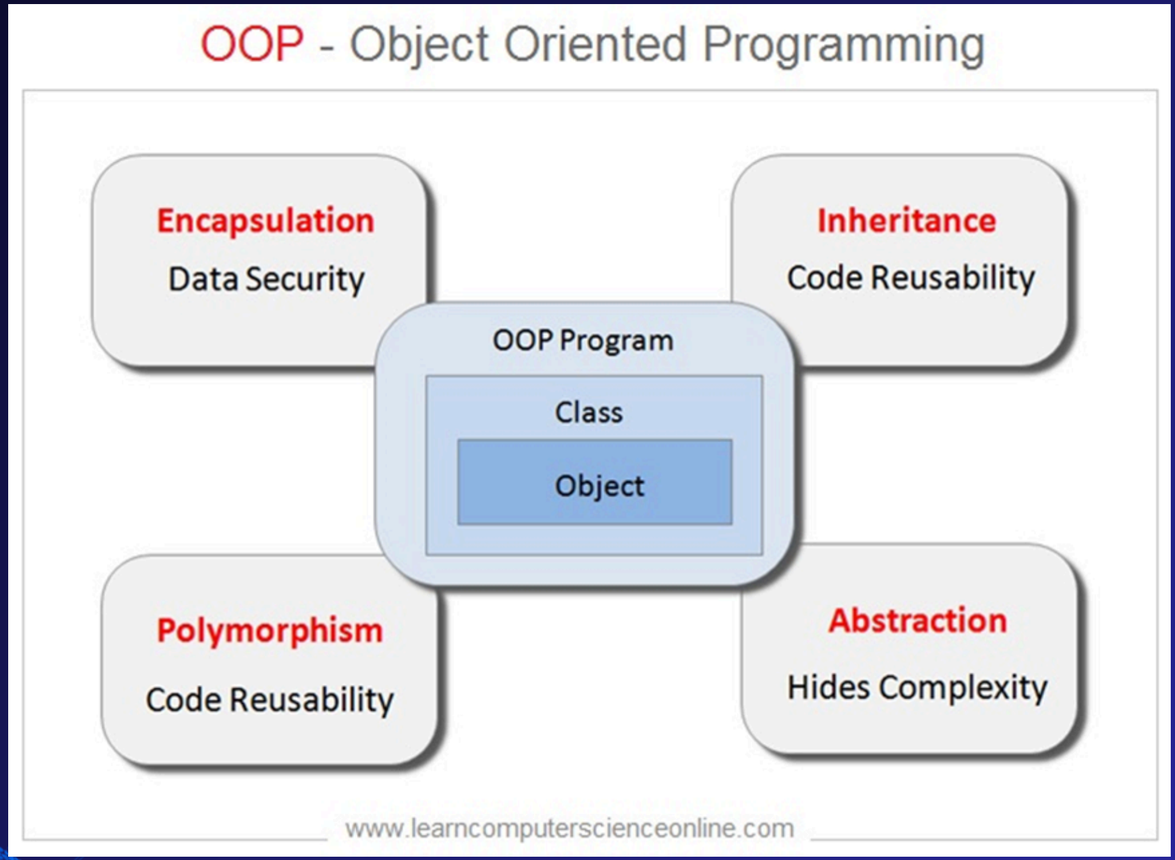
Lambda functions can have any number of arguments but only one expression, which is evaluated and returned. They are commonly used for short, simple operations, especially as arguments to higher-order functions like map, filter, and sorted.

lambda arguments: expression

```
add = lambda x, y: x + y
print(add(5, 3))  # Output: 8
```

```
numbers = [1, 2, 3, 4]
squared = list(map(lambda x: x**2, numbers))
print(squared)  # Output: [1, 4, 9, 16]
```

OOPS


OOP - Object Oriented Programming
Encapsulation
Data Security
Inheritance
Code Reusability
OOP Program
Class
Object
Polymorphism
Code Reusability
Abstraction
Hides Complexity
www.learncomputerscienceonline.com

OBJECT ORIENTED

in Python object-oriented Programming (OOPs) is a programming paradigm(pattern) that uses objects and classes in programming.

an object is any entity that has attributes and behaviors. For example, a parrot is an object. It has
attributes - name, age, color, etc.
behavior - dancing, singing, etc.

CLASS AND OBJECTS

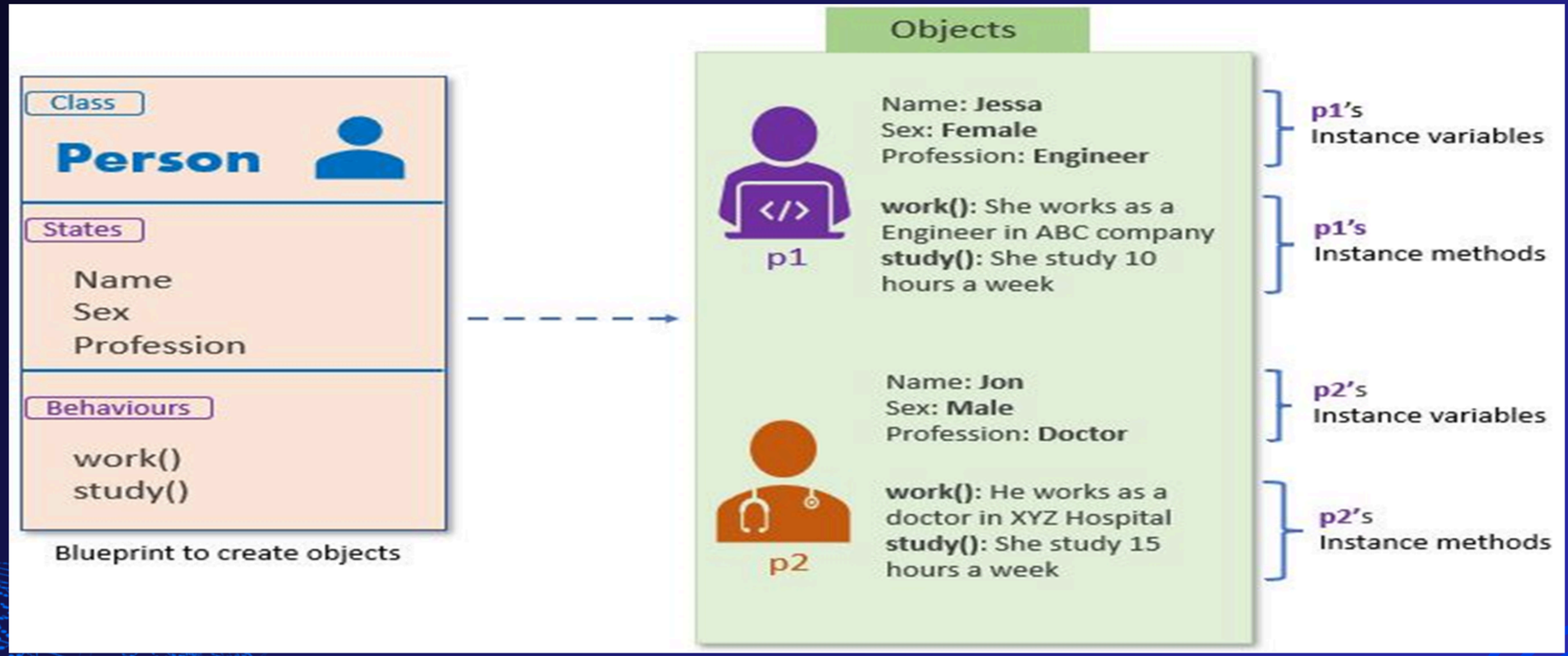
Class: The class is a user-defined data structure that binds the data members and methods into a single unit. Class is a **blueprint or code template for object creation**. Using a class, you can create as many objects as you want.

Object: An **object is an instance of a class**. It is a collection of attributes (variables) and methods. We use the object of a class to perform actions.

State(attributes): An object has an attribute that represents a state of an object, and it also reflects the property of an object.

Behavior: An object has methods that represent its behavior.

CLASS AND OBJECTS



A REAL-LIFE EXAMPLE OF CLASS AND OBJECTS.

Class: Person
State: Name, Sex, Profession
Behavior: Working, Study
Using the above class, we can create multiple objects that depict different states and behavior.
Object 1: Jessa
State:
Name: Jessa
Sex: Female
Profession: Software Engineer
Behavior:
Working: She is working as a software developer at ABC Company
Study: She studies 2 hours a day

A REAL-LIFE EXAMPLE OF CLASS AND OBJECTS.

Object 2: Jon

State:

Name: Jon
Sex: Male
Profession: Doctor

Behavior

Working: He is working as a doctor
tudy: He studies 5 hours a day

CLASS AND OBJECTS

SYNTAX:

```
class class_name:

    statement 1
    statement 2
    statement 3
```

CLASS AND OBJECTS

SYNTAX:

```
class Person:
    def __init__(self, name, sex, profession):
        # data members (instance variables)
        self.name = name
        self.sex = sex
        self.profession = profession

    # Behavior (instance methods)
    def show(self):
        print('Name:', self.name, 'Sex:', self.sex, 'Profession:', self.profess

    # Behavior (instance methods)
    def work(self):
        print(self.name, 'working as a', self.profession)
```

INHERITANCE

One of the core concepts in object-oriented programming (OOP) languages is inheritance. It is a mechanism that allows you to create a hierarchy of classes that share a set of properties and methods by deriving a class from another class. Inheritance is the capability of one class to derive or inherit the properties from another class.

INHERITANCE

Python Inheritance Syntax
The syntax of simple inheritance in Python is as follows:

```
Class BaseClass:
    {Body}
Class DerivedClass(BaseClass):
    {Body}
```

INHERITANCE

```
    # __init__ is known as the constructor
    def __init__(self, name, idnumber):
        self.name = name
        self.idnumber = idnumber

    def display(self):
        print(self.name)
        print(self.idnumber)

# child class
class Employee(Person):
    def __init__(self, name, idnumber, salary, post):
        self.salary = salary
        self.post = post

        # invoking the __init__ of the parent class
        Person.__init__(self, name, idnumber)

# creation of an object variable or an instance
a = Employee('Rahul', 886012, 200000, "Intern")

# calling a function of the class Person using its instance
a.display()
```

ENCAPSULATION

Encapsulation is a fundamental object-oriented principle in Python. It protects your classes from accidental changes or deletions and promotes code reusability and maintainability. Consider this simple class definition:

API

API stands for Application Programming Interface.

An API is an acronym for Application Programming Interface, an interface that defines the interaction between different software components. Web APIs determine what exactly request is made to the component. For example We define an endpoint to get the list of the students of a particular branch. It is also used on how to make the request and their expected responses.

IMPORTANT KEY METHODS.

- GET It is most common method for get some data from component. It returns some data from the API based on the endpoint we hit and any parameter we pass.
- POST It creates the new records and updates the new created record in the database.
- PUT It takes the new records at the given URI. If the record exists, update the record. If record is not available, create a new record.
- PATCH It takes one or more fields at the given URI. It is used to update one or more data fields. If the record exists, update the record. If the record is not available, create a new record.
- DELETE It deletes the records at the given URI.

DJANGO REST FRAMEWORK?

- Django Rest Framework (DRF) is a package built on the top of Django to create web APIs. It provides the most extensive features of Django, Object Relational Mapper (ORM), which allows the interaction of databases in a Pythonic way.
- Hence the Python object can't be sent over the network, so we need to translate Django models into the other formats like JSON, XML, and vice-versa. This process is known as serialization, which the Django REST framework made super easy.


THANK
YOU