UNIT 1

Introduction of python

PYTHON

Python is a case sensitive, high-level programming language created by Guido Van Russom in 1991, it is interpreted, interactive and object-oriented programming language.

Features of Python

* Python is *easy to learn, read and maintain*.
* Python is an *object-oriented language*- it can implement all concepts of OOPs, such as-

Encapsulation, abstraction, inheritance, polymorphism , etc.

* It is *portable or cross platform language* i.e., we can use python in several platforms like Windows, Mac, Linux etc.
* *It is free and open source:* all libraries and utilities are available free of cost, it is also an opens source which means user can find bugs, recommend solutions and even suggest updates.
* *It is extendable*, Python can easily integrate with other languages like C, java, C# (Cython, jython, ironpython
* Python has *broad standard library*, huge community support makes available broad range of libraries for python like,

Tensor Flow: For machine learning

Django: use for web development

Panda: use for data manipulation

Keras: works on artificial neural networks.

* Python provide interface to all major data base
* Python support GUI interface

History of python

Python was conceptualized by Guido Van Rossum in late 1980’s at 7CWI in Netherlands,

Name python was inspired by a BBC famous show “Monty Python’s Flying Circus” in late 1970’s.

First version 0.9.0 released in Feb. 1991

Python1 -> 1997-2000

Python 2-> 2006- 2010

Python 3-> In present

Application of python

Python is using in many applications domains mention below:

1. Web development python is good choice for web development as it offers.
   1. Frameworks such as [Django](http://www.djangoproject.com/) and [Pyramid](http://www.pylonsproject.org/)
   2. Micro-frameworks such as [Flask](http://flask.pocoo.org/) and [Bottle](http://bottlepy.org/)
   3. Advanced content management systems such as [Plone](http://www.plone.org/) and [django CMS](https://www.django-cms.org/).
2. Game development, Python is also used in the development of [interactive games](https://www.edureka.co/blog/python-turtle-module/). There are libraries such as PySoy which is a 3D game engine supporting Python 3, [PyGame](https://www.edureka.co/blog/pygame-tutorial) which provides functionality and a library for game development. Games such as Civilization-IV, Disney’s Toontown Online, Vega Strike etc. have been built using Python.
3. AI and Machine Learning, python is widely used for AI and machine learning and considered as best choice because of its wide libraries support and simple syntax

Some widely used libraries are:

Pandas, skitlern, numpy, SciPy etc.

1. Software development, python is also use for software development, most common example is scions, which is used for build control.
2. GUI Development, python is also used for development of GUI based application, Kivy is used for developing multi touch application

Python programming life cycle

Python’s programming cycle is dramatically shorter than that of traditional programming cycle. In Python, there are no compile or link steps.

Python programs simply import modules at runtime and use the objects they contain. Because of this, Python programs run immediately after changes are made.

In cases where dynamic module reloading can be used, it is even possible to change and reload parts of a running program without stopping it at all.

Python programming life cycle

Other programming language life cycle

Python IDEs

An integrated development environment (IDE) is a software application that helps programmers develop software code efficiently. It increases developer productivity by combining capabilities such as software editing, building, testing, and packaging in an easy-to-use application.

Following are some examples of IDEs.:

1. **IDLE**- When we install Python, IDLE (integrated development light environment) is also installed by default. This makes it easy to get started in Python. Its major features include the Python shell window (interactive interpreter), auto-completion, syntax highlighting, smart indentation, and a basic integrated debugger.

IDLE is a decent IDE for learning as it's lightweight and simple to use. However, it's not for optimum for larger projects

Working with python in IDLE

After installation of python software, we can work with python language in 2 different modes.

1. INTRACTIVE MODE also known as command line mode, here we give one command at a time and python also executes that command instantly.
2. PROGRAMMING MODE also known as script mode, here we write whole program first then and save the program with extension .py
3. PyCharm Jet Brains is the developer, PyCharm also provide error detection and remediation
4. Spyder Spyder is an open-source Python IDE that’s optimized for data science workflows. Spyder comes included with the Anaconda package manager distribution, so depending on your setup you may already have it installed on your machine.
5. Tonny Thonny is an IDE for Python 3, it has simple user interface.
6. PyDev as an external plugin for Eclipse, PyDev is one of the most popular Python IDEs. The Python programmers who have a background in Java naturally gravitate towards this Python interpreter because it is so well-liked by users.
7. Jupiter notebook Jupyter is one of the most used Python notebook editors that is used across the Data Science industry. You can create and edit notebook documents using this web application, which is based on the server-client architecture. It utilizes Python's interpretive nature to its fullest potential.
8. Google Collab It gives free GPU access. Instead of downloading heavy files and tedious launch time, one can directly update the files from Collab to the drive.

Some text editor also supports Python:

Sublime Text Notepad++

Atom Vim

Visual Studio Code

Token are the set of legal character use in that language. Python have following token. There are total 35 keywords in Python 3.11 .

Keywords, identifier, literal, operator, Punctuation.

**Keyword** keywords are the reserve words which have some specified meaning for the interpreter

Example-

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| False | True | None | And | or |
| not | With | Def | Return | lambda |
| If | Elif | Else | For | While |
| Pass | Break | Continue | try | Finally |
| Assert | Except | raise | As | from |
| import | Yield | Global | is | in |

Code:

import keyword

print(keyword.kwlist)

Identifiers also known as user defined words, these are name given by user to different variables, object, function name or class name, to any other data type like list, tuple etc.

Rules for writing identifier

* It contains alphabets (A-Z or a-z), digits (0-9) and special symbol (underscore \_).
* No white space or blank space is allowed.
* Reserve words cannot be used as identifier name
* First character of an identifier should be start with character or an underscore (\_)

Some naming conventions.

* Camel Case: Eg: firstName
* Snake Case: first\_name
* Screaming snack case: FIRST\_NAME
* Pascal’s case \_first\_name\_

In python there are some variable which start from double underscore and also ends with double undersore. Such variables are known as dunder variable

Literals a fixed numeric or non-numeric value is known as literals, it can define as a number, text or other data that represents values to be stored in variable.

Types of literals

String Literal, Numeric Literal, Floating literal, Boolean Literal, Collection Literal, (c= [23,45,67,88])

String literals: Anything written in quotes is known as string literal in python

e.g., a =” Hello”,

Numeric Literals: numeric literal consist digits, with symbol + & - . python supports int, float and complex numbers.

Example: b= 24,

Boolean Literal: Boolean literals supports only two values, “TRUE” & “FALSE”

Operators operator is symbol that performs some operation on data, the data on which operation will performed is known as operand

Different types of operator.

Arithmetic operator It performs arithmetic calculations on numeric literals

|  |  |
| --- | --- |
| Symbol | Description |
| + | Addition |
| - | Subtraction |
| \* | Multiplication |
| / | Division |
| % | Reminder |
| \*\* | Exponent |
| // | Floor division |

Relational operator operators used for comparisons purpose i.e., tells the relation between operands

|  |  |
| --- | --- |
| Symbol | Description |
| < | Lesser than |
| > | Greater than |
| <= | Less than equals to |
| >= | Greater than equals to |
| == | Equals to |
| != | Not Equal to |

Logical operator

|  |  |
| --- | --- |
| Symbol | Description |
| and | Logical AND |
| or | Logical OR |
| not | Logical Not |

Assignment operator uses to assign values to variable

|  |  |
| --- | --- |
| Symbol | Description |
| = | Assignment |
| += | a+=2 : a= a+2 |
| -= | a-=2 : a= a-2 |
| \*= | A\*=2 : a= a\*2 |
| /= | a/=2 : a= a/2 |
| %= | A%=2 : a= a%2 |
| \*\*= | A\*\*=2 : a= a\*\*2 |
| //= | a//=2 : a= a//2 |

Bitwise operators: Operators which perform operations over bits are known as bitwise operators

|  |  |
| --- | --- |
| Symbol | Description |
| & | Bitwise AND |
| | | Bitwise OR |
| ~ | Bitwise Not |
| ^ | Bitwise XOR |
| >> | Right Shift |
| << | Left Shift |

Identity operator These operators compare two objects for equality and return True or False as a result

|  |  |
| --- | --- |
| Symbol | Description |
| is | Are the two objects same |
| is not | Are the two objects same |

Membership operator it checks the presence of an object in the sequence and return True and False as a result

|  |  |
| --- | --- |
| Symbol | Description |
| in | Is object present in the object |
| not in | Is object not present the object |

Operator precedence

Python operators have a set order of precedence, which determines what operators are evaluated first in a potentially ambiguous expression.

Higher priority order operator will be evaluated first then low priority

|  |  |
| --- | --- |
| Symbol | Description |
| ( ) | Parentheses |
| \*\* | Exponent |
| \* / // % | Multiplication/ Division/ Modulus |
| + - | Addition/ Subtraction |
| << >> | Bitwise Shift Left & Bitwise Shift Right |
| < <= | Relational Less / Less than or equal to |
| > >= | Relational Greater / Greater than or equal to |
| == != | Relational equal to / is not equal to |

Operator Associativity it tells the order of evaluating the operator of same precedence

There are 2 types of associativity

1. Left to right
2. Right to left

Python have left to right associativity

Punctuations, punctuations are the symbols that are allowed in the programming language that use to write, organize, and format statements, expression and program structure. Python supports following punctuations-

( ) { } [ ] ‘ “ # , @

Python block structure

A group of statements are known as block and in python we use indentation to defines a block.

For indentation user may use 4 spaces or 1 tab. User must not combine the space and use of tabs it may cause syntax error

Python block structure

Block 1 Start

Block 2 Starts

1 Tab or 4 spaces

Block 2 Ends

Block 1 Ends

Comments A comment is a part of the code that the programmer does not want to execute, rather the programmer uses it to either explain a block of code or to avoid the execution of a specific part of code while testing.

Single line comment:

#this is single line comment

Multiple line comment:

‘‘‘ Triple single inverted commas are used for multiline commenting ’’’

Print function

For printing information on the screen, we use print( ), its automatically appends the new line character after printing information on screen

print(“prints as a message”)

print(<values of variable>)

format printing, with the help of formatting we print the output in more readable form.

Syntax:

print ( f”{<variable\_wants to print>} ” )

Example:

A= “Ram”

Print(f”Hello { A }”)

o/p:

Hello Ram

Escape character: These are the sequence start with a slash are known as escape characters , with the help of these we can prints some unprintable characters

Example:

|  |  |
| --- | --- |
| Escape character | Uses |
| \\ | Prints a backslash (/) |
| \t | Prints a tab |
| \’ | Prints a single quote |
| \b | Prints a backspace |
| \n | Prints new line |

Input function input function use to get data from user, it accepts input in the form of string , if we want to take input as different data type then we have to type cast it into that data type. We must store the input value a variable for future use.

Syntax:

<Variable> = input(“ Message shows on output screen”)

Example:

Name = input(“Enter your name: ”)

print(Name)

o/p:

Typed from output screen, while program in running phase

Enter your name: Shubhi

Shubhi

Data types: Python have 4 types of data type.

1. Numeric: int, float, complex.
2. Sequence Type: String, List, Tuple and Set.
3. Boolean Type: True and False.
4. Dictionary Type

In python data types can be divide into two categories : Mutable and immutable

**Mutable data type:** Value of mutable data types can be changed after creation.

**Examples:** List, Set, Dictionary

**Immutable:** Data types whose value does not change after it once created.

**Example:** Tuple, String.

We use type function to check the data type of any variable

Syntax: type(<variable\_name>)

Example:

A=”Hello”

type(A)

O/P:

<class 'str'>

List: List is mutable data structure which stores heterogenous data types like: numeric, string, list, tuple, dictionary, Boolean.

List is written in square brackets [ ] & different elements are separated by comma(,).

Syntax:

Variable\_name = [item 1, item 2, item 3, item 4, ……….. , item n]

Example:

Fruit = [“Apple”, “Banan”, “Grapes”, “Mango”]

a=[ ] or a= list( ) # creates an empty list with name “a”.

We can create the list dynamically via keyboard using eval( ) function or list( ) function to create a list dynamically.

O/P:

enter your list: [1,2,3,4,5]

[1,2,3,4,5]

<class 'list'>

Example:

a=eval(input(“enter your list: \t”))

print(a)

type(a)

or

O/P:

enter your list: 12345

[1,2,3,4,5]

<class 'list'>

a=list(input(“enter your list: \t”))

print(a)

type(a)

Print whole list:

print(<list name>)

O/P:

[1,2,3,4,5,6]

Example:

a=[1,2,3,4,5,6]

print(a)

print(<list name>)

O/P:

++1,2,3,4,5,6

Example:

a=[1,2,3,4,5,6]

print(\*a)

Excess elements:

Syntax:

<list\_name>[<index of parents list elements>][“index of child list element ”]

List Slicing: List slicing is the process of using particular portion of list.

Syntax:

List\_name [ initil : end : index jump ]

Example:

Fruit = [“Apple”, “Banana”, “Grapes”, “Mango”]

print( Fruit [1:3])

O/P: Banana, Grapes.

Python have Positive and negative indexing, where -1 represents last elements of a sequence.

OPERATORS:

+ for concatenation, combines two list

\* Replication, copy the list multiple times.

We can also use membership operator to check the availability of an element in the list.

BUILT\_IN Methods

1. len(<list\_name>) 🡪 calculates the length of list
2. min(<list\_name>)🡪 prints the minimum value
3. sum( <list\_name>) 🡪return the sum of all elements in the list
4. max(<list\_name>) 🡪 return the maximum no in a list
5. reversed(<list\_name>) 🡪 Reverse the content of list : needs list type conversion
6. <List\_name>.sort() 🡪 Rearrange the elements in ascending order.
7. <list\_name>.reverse() 🡪 Reverse the all elements of list .
8. <list\_name>.insert(index,element)
9. <list\_name>.append(element) 🡪 Add element at the end of list
10. <list\_name>.clear() 🡪 Remove all the elements from the list
11. del <list.name>[a:b] 🡪 Remove sliced elements.
12. <list\_name>.remove(element) 🡪 remove the element from the list
13. <list\_name>.pop() 🡪 Remove and return the last element of list, may also pass index as an argument to remove element from the list.
14. list()🡪 convert any data structure into list.

‘ We can also create nested list which means list consist of another list.

Eg:a= [[“ABC”, “24”,”F”],[“ xyz”, “25”,”M”],[“ MNO”, “23”,”F”]]

Tuple: Tuple is immutable and an ordered collection of elements enclosed in parenthesis ( ), once you create a tuple you cannot change the value inside of tuple.

Syntax:

<tuple\_name>=(item 1, item 2, item3, …………., item n)

Example:

tup=( “abc”, 100, True, 4.5)

tup=( ) #emplty tuple

tup=(1,) #tuple with single element.

We can also create tuple dynamically using tuple( ) or eval ( ) function .

Example:

Example:

O/P:

enter your list: (1,2,3,4,5)

(1,2,3,4,5)

<class 'tuple'>

a=eval(input(“enter your tuple: \t”))

print(a)

type(a)

or

Example:

O/P:

enter your list: 123456

(1,2,3,4,5,6)

<class 'tuple'>

a=tuple(input(“enter your tuple: \t”))

print(a)

type(a)

OPERATORS:

+ for concatenation, combines two tuples

\* Replication, copy the tuple multiple times.

in, not in : membership operator

Functions of tuple:

1. otuple.count(item): counts the frequencies of item in tuple.
2. all(tuple): it return True if all the elements are true
3. len(tuple): returns the length of tuple.
4. max()

String:

String is an immutable data structure, use to store set of characters in inverted quotes.

In python we can use ‘single inverted comma ’, “ double inverted comma ”, ‘’’ triple single inverted comma ‘’’, “”” triple double inverted comma “””.

Triple inverted commas are used to store multi line strings.

OPERATORS:

+ for concatenation, combines two tuples

\* Replication, copy the tuple multiple times.

in, not in : membership operator

Some Built-in functions

1. <String\_name>.lower() 🡪 Converts the string in lower case.
2. <String\_name>.upper() 🡪 Converts the string in upper case.
3. <String\_name>.title() 🡪 Capitalized first letter of each word .
4. <String\_name>.capitalized() 🡪 Capitalized only first letter of sentence.
5. <String\_name>.swapcase() 🡪 swap the lower and upper case.
6. <String\_name>.endswith(character) 🡪 Converts the string in lower case.
7. <String\_name>.find(“sub string”) 🡪 Tells the index of sub string.
8. <String\_name>.endswith(character) 🡪 Converts the string in lower case.
9. <String\_name>.replace(“old string”, “new string”) 🡪 Replaces the old the string with new string.
10. <String\_name>.strip(character) 🡪 Removes both leading and trailing whitespace from the string .
11. <String\_name>.center(spaces) 🡪 Add leading and trailing whitespace .
12. <String\_name>.split(spaces) 🡪 Add leading and trailing whitespace .

Sets and dictionary are typical examples of unordered and unorganized collection

Set: Set is the collection of unique elements, it is un ordered and un index data enclosed in curly brackets { }.

set\_name = {item 1, item 2, ………… , item n}

Example:

S={1,2,3,4}

s= set() #Emplty set

Operations on set

1. Union ( | ): Example:

O/P:

{1,2,3,4,5}

a= {1,2,3}

b={2,3,4,5}

union = a|b

print(union)

1. Intersection ( & ): Example:

a= {1,2,3}

O/P:

{2,3}

b={2,3,4,5}

inter=a&b

print(inter)

1. Difference ( - ): Example:

a={1,2,3}

b={3,4,5}

O/P:

{1,2}

difference = a-b

print(difference)

1. Symmetric difference:

a={1,2,3}

O/P:

{1,4,5}

b={2,3,4,5}

sd= a^b

print(sd)

1. Membership operators: in & not in

Some Built-in functions

1. len(set\_name) 🡪 Tells the length of set.
2. <set\_ name >.add(element) 🡪 Add a element to the set
3. <set\_ name >.clear( ) 🡪 Delete all elements from the set
4. <set\_ name >.pop( ) 🡪 It removes any item from the set, it throws keyerror if set is empty.
5. <set\_ name >.remove(item ) 🡪 Delete the given item from the set
6. <set\_ name >.union(new\_set ) 🡪 Return new\_set contain all elements of set\_ name
7. <set\_ name >.intersection(new\_set )
8. <set\_ name >.difference(new\_set )
9. <set\_ name >.Symmetric\_difference(new\_set )

Dictionary:

A dictionary consists of key and values also known as mapping in python. In dictionary instead of indexed numerically like last, it has keys, values within the dictionary. Values can be accessed by keys. We can store all the data types in dictionary.

Syntax:

<Dict\_name> = {key1: value 1,

Key2: value 2,

:

.

:

.

}

Example:

S\_marks = {“Harsh”: 24, “Tushar”: 48}

D = { } # empty dictionary

We can also create dictionary using dict( ) method.

Example:

O/P:

{1: 'hello', 2: 'hey', 3: 'hi'}

greet=dict([(1, 'hello'), (2, 'hey'), (3, 'hi')])

print(greet)

Access Dictionary:

print(<dict\_name>[<key\_value>]) #prints value of given key

print(<dict\_name>) #prints complete dictionary

print(\*<dict\_name>) #prints only keys of dictionary

Assign values:

<dict\_name>[<key\_name>]=<value>

Built in functions in dictionary

1. Len(<dict\_name>) 🡪 Return length of dictionary
2. <dict\_name>.keys() 🡪 prints all keys present in dictionary.
3. <dict\_name>.values() 🡪 prints all values present in dictionary.
4. <dict\_name>.items() 🡪 prints all key-value pair present in dictionary in form of tuple
5. <dict\_name>.clear() 🡪 Removes all the elements from the dictionary.
6. <dict\_name>.fromkeys(seq, val=None) 🡪 creates a dictionary with element of sequence as keys.
7. <dict\_name>.get(key) 🡪 returns the value of key if it found otherwise return None.
8. <dict\_name>.pop(key) 🡪 remove given key and return its value in dictionary.
9. <dict\_name>.popitem() 🡪 Remove the last key-value pair and return as tuple.
10. <dict\_name>.update(<dict\_2>) 🡪 merge two dictionaries.

Type casting

Converting one data type into another is known as type casting or type conversion.

Type casting is of two types: Implicit and explicit.

* In implicit type conversion interpreter converts the data type automatically.
* In explicit type conversion programmer converts the data type by its choice.

Following are the methods used for explicit type casting:

1. list(<variable\_name>) #converts a data type into list
2. tuple(<variable\_name>) #converts a data type into tuple
3. set(<variable\_name>) #converts a data type into set
4. int(<variable\_name>) #converts a data type into integer
5. str(<variable\_name>) #converts a data type into string

UNIT 2

Python program flow control conditional block

Conditional Statement

If Statement:

“if” is a simple way of checking condition if the coditio is true than the set of statement will be execute, which were written in indentation

Condition ?

False

True

Statements

**Syntax**:

If(condition):

Statement

Indentation

**Example**:

Code:

O/P:

a is +ve

a=6

if a>0:

print(“a is +ve”)

Else :

When if statement is not true then else block will run.

**Flow diagram:**

Statements

False

Condition ?

True

Statements

Statements

**Syntax:**

else: #No condition is required for else statement

<statements>

**example:**

Code:

O/P:

a is -ve

a=-3

if a> 0:

print(“a is +ve ”)

else:

print(“a is -ve”)

even if a is equal to 0 then also it prints a is -ve.

elif: elif is used to check condition before going to else statement.

Flow chart:

False

Condition ?

Condition ?

False

True

True

Statements

Statements

Statements

Statements

Syntax:

elif <condition>:

<statements>

Example:

Code:

a=0

O/P:

A is equal to Zero

if a> 0:

print(“a is +ve ”)

elif a==0:

print(“a is equal to Zero”)

else:

print(“a is -ve”)

Ternary expression:

**Syntax:**

<do this> if <condition is true> else <do this>

Example:

O/P:

a is even

Code:

Print(“a is even”) if a%2 ==0 else print(“a is odd”)

Questions

1. Write a python program to calculate.

F(x)= x3+5x+3 if x<=2

x+3 if x>2

1. Check whether the candidate is eligible for voting or not.
2. Write a program to calculate electricity bill.

|  |  |
| --- | --- |
| Unit | Price |
| First 100 unit | Free |
| Next 100 units | Rs. 5 per unit |
| After 200 units | Rs.10 per unit |

1. Check whether the year is leap year or not.
2. Write a program to display tourist places of different city.
3. Check whether the character is vowel or not.
4. Write a program for arithmetic calculator.

Loop in python.

Sometimes we want to repeat a set of statements in our program for this we use loop to make our code shorter and easier. For example we want to print numbers from 1-100.

In python there are two types of loops.

1. While Loop
2. For Loop

While Loop

A while loop executes the statements as long as the given condition is true.

**Syntax:**

while <condition>:

#code

Increment/decrement

**Example:**

O/P:

1

2

3

4

.

.

10

Code:

i=1

while 1<10:

print(i)

i+=1

For Loop:

In python ‘for’ loop iterates over a collection of items such as list, dictionary etc.

**Syntax:**

for <variable> in <sequence>:

#code

**Example:**

O/P:

1

2

3

4

5

6

Code:

a =[1,2,3,4,5,6]

for i in a:

print(i)

we can use built-in function range( ) which returns sequence from 0 to given argument.

**Example:**

O/P:

0

1

2

3

Code:

for i in range(5):

print(i)

we can use slicing

**syntax:**

for i in range(start, stop, step):

#code

E**xample:**

O/P:

1

3

5

7

9

Code:

for x in range(1,11,2):

print(x)

Questions

1. Print prime numbers from 1-100.
2. write a program to print table of 5.
3. write a program to print table of 5 in reverse order.
4. Calculate the sum of numbers from 1 to 10 using a for loop
5. Calculate the product of elements in a list using a for loop
6. Print even numbers from 1-10
7. Print each character of a string.
8. Print factorial of a number.

Jump Statements:

Jump Statements are the statement use to hinders the flow of code. Python supports two jump statements: Break, continue.

Let’s understands them in details

**Break**: Berak statement is use to exit from the loop body immediately also known as abnormal loop termination or premature loop exit.

**Flow Diagram:**

False

Condition ?

True

True

Break

Condition ?

False

Statements

Statements

Example:

Code:

O/P:

0

1

.

.

30

for i in range(50):

print(i)

if i ==30:

break

**Loop with else clause**

In python we can also use “else” clause with loop statements and it executes when loop terminates normally, i.e., without encountering the break statement.

**Flow Diagram**

False

Loop Condition?

Else:

Statements

True

Statements

**Example:**

Code:

a=[5,3,13,17]

for i in a:

if i%2==0:

break

else:

print(“given list consist odd numbers”)

**Continue**

It is use to executes the loop from start and by skipping the remaining part/ statements of the block.

**Flow Diagram**

False

Loop Condition?

True

Continue Statement

True

Statements

False

Statements

Example:

for i in range (1,11):

if i%2==0:

continue

print(i)

**Pass** This statement does nothing i.e., Null Operation. Whenever we write ‘pass’ interpreter comment outs that block.

**Flow Diagram:**

Block Start

False

True

Statements

Pass Statements

Example:

print(“Top of Loop”)

for i in range(1,2):

pass

print(“Bottom of Loop”)

Patterns Program in python

1. Square pattern.
2. Right angle triangle
3. Right right angle triangle
4. Pyramid
5. Diamond

Unit 3

Function Paradigm

Functional Programming (FP) is a programming paradigm. A paradigm is a framework that indicates way of computational thinking requires to solve a given problem.

Functions are the structured or procedural programming way of organizing the logic in program. Large blocks of code can be neatly segregated into manageable chunks. So, functions are the units of program which performs a particular task, take some input and may return some output.

**Features:**

1. Modular Programming: Division of code into functions make design efficient.
2. Reusability of code (no redundancy)
3. Manageability
   1. Easy debugging
   2. Efficient
4. Parallel development
5. Smaller code
6. Specifies things: One function is created for specific use.

**Terminology:**

1. Name of function: what is the name of function which we want to use or create.
2. Arguments: inputs given to the function to process inside the body of function.
3. Return value: value return to the main code after executing the function. We need to store that return value in a variable.

There are two types of functions based on return values: Void function and Fruitful function.

**Void Function** is a function which does not returns a value, it executes the statements perform the task but does not return value.

**Fruitful Function** is a function which returns a value after exacting the body of function.

Functions can be categorized into two groups built-in and user define function

User define Function:

Functions which are created by programmer as per its requirement are known as user define function

There are two components of function.

1. Function definition
2. Function calling: we call the function just by writing function’s name.

**Syntax:**

def <name\_of\_function> (list of parameters):

#code/ body of function

**Example:**

def greet(): #function definition

O/P:

Hello dear user

print(“Hello dear user”) #function body

greet() #function calling

**Argument and Parameter.**

Arguments are the Input values that are passed from function call to the function values.

While parameters are the placeholders for input use inside the body of function.

a=5

Parameters

b=6

def sum(x,y):

print(x+y)

sum(a,b)

Arguments

**Function with return value**

def sum(a,b):

return a+b

a= greet(4,5)

print(a)

Types of function arguments

1. Function with no arguments.
2. Functions with positional arguments
3. Function with keyword or named arguments
4. Function with default arguments
5. Functions with variable key and keyworded arguments
6. **Function with arguments**

def greet (name):

print(“hello”, name)

greet(“Shubhi”)

1. **Function with positional arguments**.

In this type of arguments first arguments stores in first parameter second in second and so on. We generally use these arguments in our program.

1. **Function with keyword arguments**

In this type of arguments we assign value directly to the arguments .

def person(name, age):

    print("Hi, I am", name)

    print("My age is ", age)

person(name="john", age=20)

person(age=20, name="Prince")

1. **Function with default arguments**

In these types of function we give some default values to the parameters.

We can call a function without passing any argument, then default values can be use for future code. If we assign values to the arguments then given values use for values.

def sum(a=5,b=6):

print(a+b)

sum()

1. **Function with variable length arguments**
2. **\* Arguments:** In python we can store arbitrary number of arguments in a function in the form of tuples, via adding \* as a prefix of parameter.

def person(name,\*data):

print(“Name is:”, name)

print(“have marks”, data)

person(“John”, 45,24,46,78)

1. **\*\* Arguments** : It allows us to pass dictionary as a arguments to the calling function.

**Eg.**

def person(name,\*\*data):

print(“Name”, name)

print(“Data of the user”, data)

person(“John”, age=24, mobile=91xxxxxxxxxx)

**Anonymous Function or Lambda function**

Lambda function is a small inline function which can take any number of arguments but only one expression. The parameters of the lambda are define on the left side of colon and body of function is define on the right side of colon result of the body will implicitly returned

Syntax:

<Identifier>= lambda <list of parameters> : <expression>

Example:

s= lambda a,b: a+b

print(s(8,9)) => o/p: 17

Some built in function:

1. map function( )- This function is use for applying a function on each element of iterable. So it takes two formal arguments a function and item collection.

Syntax:

<identifier> = map(function, iterable)

Example:

a=[1,2,3,4,5]

def s(element):

return element\*\*2

lis=map( s, a)

print("List before Map Function", a) =>O/P: List before Map Function [1, 2, 3, 4, 5]

print("List after map function",list(lis)) =>O/P: List after map function [1, 4, 9, 16, 25]

1. filter function: This function filters the value of iterable, it takes a function and a iterable and returns the item of iterable if function returns True.

We need to typecast the output generated by the function filter

Syntax:

<identifier> = filter(function, iterable)

Example:

a=[1,2,3,4,5,6]

f=filter(lambda x: x%3==0, a)

print("After applying filter function",list(f)) => [3,6]

Modules: Modules are general files that contain set of functions, classes and methods each doing its own function to achieve an end goal.

Advantage:

* Reusability
* Easy to use
* Scope- in a single program user can invoke multiple modules.

To use a module in a program we need to import that module in our program

Syntax:

1. import <module\_name>
2. from <module\_name> import <func1, func2….>
3. from <module\_name> import \*

Module Aliasing:

We can import a particular module and refer it with an other name of our own choice using a keyword ‘as’.

Syntax:

import <module\_name> as <name\_by\_programmer>

example:

import math as m

Modules can be user define or built-in or user define

Some built-in modules

1. math – math.factorial(), math.floor(), math.gcd(),……..etc
2. cmath – mathematical function with complex number
3. decimal – sqart(), exp(),ln(), log10(),…..etc.
4. random – It generates random values
5. statistical module- mean(), median(), mode(),…..etc
6. re – use for regular expression

and so on

example:

import random

print(random.random( )) => return float no b/w 0&1

print(random.randint(10,55)) => return random no. b/w given range

For user define module we can use any python program as a module in any other program with same method

We can use “if \_\_name\_\_ = = \_\_main\_\_” for mention that the code under this section will only run if we are running this file as a main not as a module.

Zen of Python

The Zen of Python is a collection of 19 "guiding principles" for writing computer programs that influence the design of the Python programming language. Software engineer **Tim Peters** wrote this set of principles and posted it on the Python mailing list in 1999.

It displayed by typing “import this”.

1. Reduce function: Reduce function takes a function and iterable as a formal arguments and return a single value by combining the elements of iterable.

Syntax:

<identifier> = reduce(function, iterable)

Example:

from functools import reduce

total = reduce(lambda a, x: a + x, [0, 1, 2, 3, 4])

print(total) => 10

Variable scope

Scope of an identifier is defined to be the portion of code where we can access it. Variables either have local or global access.

Scope are of two types

1. Global Scope – Names which we can use globally i.e., throughout the program are known as global scope variable
2. Local Scope – Names which we can use only in particular block are known as local scope variable.

Example:

a=[1,2,3]

def change(n):

a=[6,7,8]

print(a) => [6,7,8]

change(a)

print(a) => [1,2,3]

NameSpace – A namespace is a collection of currently defined symbolic name along with information about the object each name reference. It contains name of variables. It ensures that all the names are unique.

Types of Namespace

1. Built-in Namespace - Built-in Namespace contains th3e name of Python’s built-in objects which are available to the interpreter all the time. This namespace starts when the program startup and exist until the terminator exists.

Code:

dir(\_\_builtins\_\_) => print all built-in namespace

1. Global Namespace – Global namespace is created when the main body starts and terminates with the termination of interpreter. It also create a namespace of all the module that our program imports.
2. Local or Enclosing Namespace – Local namespace is define when a function is call, each function has its own namespace and it deletes when function is terminates.

Enclosing namespace works when we use nested function and it get activated when parent function call.

Python complex data types

List comprehension

List comprehension creates a new list by applying an expression to each elements of an iterable.

Syntax:

<variable\_name > = <expression> **for** <element> **in** <iterable>

Example:

O/P:

[16, 25, 36, 49, 64]

Square=[ x\*x for x in (4,5,6,7,8)]

print(Square)

we can also use if condition to filter values e.g., print the square of only even values.

Syntax:

<Variable\_name> = [<expression> **for** <element> **in** <iterable> **if** <condition>]

Example:

Even = [x\*x for x in [2,3,4,5,6,7,8] if x%2==0 ]

print(Even)

In python we can also create Dictionary and Set comprehension.

Set

Example:

O/P:

{2, 100, 4, 5, 7, 9}

<class 'set'>

l=(1,3,6,8,99,4)

a={a+1 for a in l}

print(a)

type(a)

Dictionary

Example:

O/P:

{1: 1, 3: 9, 6: 36, 8: 64, 99: 9801, 4: 16}

<class 'dict'>

d={x: x\*x for x in l}

print(d)

type(d)

Questions

1. Remove the duplicate element from the list using list comprehension.
2. Write a program that takes a dictionary and returns a new list containing all the keys from the dictionary using list comprehension. Example: Input: {"a": 1, "b": 2, "c": 3}, Output: ["a", "b", "c"].

More example:

#only with filter method

from random import randint as rn

def odd(n):

return n%2

num=[]

for i in range(9):

num.append(rn(1,99))

print(list(filter(odd,num)))

#with lambda function

from random import randint as rn

num=[]

for i in range(9):

num.append(rn(1,99))

print(list(filter(lambda n: n%2 ,num)))

we can filter function with list comprehension

#with list comprehension

from random import randint as rn

num=[]

for i in range(9):

num.append(rn(1,99))

print([n for n in num if n%2])

# shorter code

from random import randint as rn

print([n for n in [rn(1,99) for i in range (9) ] if n%2])

Unit 4

Python file operation

File Handling

File handling is the process of preserving the data for future use it is also known as file processing. It is important part of web application. File we create in file processing is known as data file.

There are two types of file in python

Text file, it stores information in plain text format, we can create text file using notepad, Ms-office, Ms-word etc.

binary file, binary file use to store binary data such as image, video, audio etc. in binary file there is n+ o delimiter for a line, so these files are easier to and much faster, files other than text.

**Operation in file handling**

Basically there are 4 operation in file handling commonly known as CRUD

CRUD

Deletion

Update

Read

Creation

File handling system

Open a file

Firstly we have to open a file for work on it, in a particular mode depending on the purpose for which the file is to be open.

*We use open function for opening a file.*

Syntax:

open( “<file\_name>” , “<file\_mode>”)

We use an identifier file handle also known as file object to keep the link of file residing in our computer.

Modes

“r”- it opens a file in read mode and throws an error if does not exist. Read is ;ikthe default value of file mode.

“w” – it opens a file for writing purpose and it overwrite the content of file. Also it create the file if that file is not exists.

“a” – it opens a file in append mode i.e., here we add the content at the end of existing data . if that file does not exist it will create it .

“r+” – opens file in read with writing mode, but it throws an error if does not exist.

“w+” – opens a file in write with reading mode , it does not throw an error if does not exist rather it will create it.

“a+” – it opens a file in append with reading mode.

“x”- it will create a file and throws a FileExistError if that file is already existed.

We can specify the type of file as text and binary with mode.

t-> use for text file and it is also default value

b-> use for denote binary file.

Eg-

rb-> use for opening binary file in read mode.

wb-> use for opening binary file in write mode.

ab-> use for opening binary file in append mode.

rb+-> use for opening binary file in read + writing mode.

wb+-> use for opening binary file in writing +read mode.

ab+-> use for opening binary file in append +read mode.

Syntax:

Method 1

<file\_handler> = open( “<file\_name>” , “<file\_mode>”)

Here we have to close the file performing the work.

Method 2

with open( “<file\_name>” , “<file\_mode>”) as <file\_handler> :

#code

In this method there is no need of closing the file while coding it will automatically close it.

Method 3

In this method we use exception handling for accessing the file.

***Note***:

Whenever there is something wrong happens in the code, program will halt, we use exception handling to deal with such situations. This method works for exception handlings logics.

try:

<file\_handler> = open( “<file\_name>” , “<file\_mode>”)

finally:

<file\_handler>.close( )

Example:

f = open(“Python.txt”,”a”) # it will opens Python.txt file in append mode.

Close operation every file must be closed which is open in the program before its termination otherwise file may not updates.

Syntax:

<File\_handle>.close( )

Example:

f.close()

Read operation we perform read operation by opening a file in read mode (‘r’),

**Methods to read file**

1. *Read() -* we can pass the parameter which specify how many characters will be be read, it will read whole document by default.

Syntax:

<file\_handle>.read(size)

Example:

f = open('python.txt','r')

complete=f.read()

print('whole document')

print(complete)

f.close()

f = open('python.txt','r')

str=f.read(20)

print('first 20 bits of file')

print(str)

f.close()

2. *readline()-* readline method is used to read only one line (byte by byte) from the file at a time. However, if the number of bytes are specified then it reads only the number of specified bytes of the line from the file.

Example-

with open('python.txt','r') as f:

s=f.readline()

s1=f.readline(20)

s2=f.readline()

print(s)

print(s1)

print(s2)

1. *readlines()*- it read all the lines of the document and return them as a element of list.

*Example*-

with open('python.txt','r') as f:

s=f.readlines()

print(s)

print(s[0])

**Write Operation**

It will overwrite the whole the content on the existing of the file.

Syntax:

File = open(“file\_name”, ‘w’)

File.write(“content user wants to write”)

File.close()

Example-

File=(“text.txt”,’w’)

File.write(“No this content is overwriting this file”)

File.close()

*append ()*

*Example:*

f= open (“python.txt”,’a’)

f.write(“It’s a append text”)

f.close()

Some other functions use for file handling

Flush () – Flush function is to writing data immediately into internal memory from internal buffer. We use it when we want any delay in writing the data into a file even without closing the file.

Syntax:

<file\_object>.flush()

Example:

f = open("test.txt",'w')

f.write("also with flush method ") => there will be no content in the file

f = open("test.txt",'w')

f.write("also with flush method ")

f.flush() => write the content even without closing the file.

File Pointer

A file pointer is a marker that indicates the current position of the cursor in the file. Initially file pointer points at beginning of the file. It moves after each read or write operation.

We can randomly access the file pointer, there are two functions for randomly accessing the file: seek() and tell().

1. seek()- It is use to change the position of the file pointer at a specified location.

Syntax: <file\_handler>.seek(<position of pointer in byts>)

Example:

f=open(“python.txt”)

f.seek(4)

print(f.read()) => text from 4th index

1. tell() – It is use to know the current location of the file pointer.

Syntax: <file\_handler>.tell()

Delete Operation:

Delete operation is done by importing OS module.

Syntax:

import os

os.remove(“file\_name”)

Example:

import os

if os.path.exists(‘file\_path’): -> Checks whether file exist or not

os.remove(‘file\_name’)

print(‘File deleted successfully!’)

else:

print(‘file not exist’)

we can even delete folder to.

Syntax:

os.rename(‘old\_name’,’new\_name’) use to rename files

os.getpwd() -> return present working dir

os.chdir() -> change the directory

import os

os.rmdir(“folder\_name”)

Unit 5

Packages

A package is made up of multiple python files or module. All python package must contain \_\_init.py\_\_ .

When we import any package \_\_init.py\_\_ will be run and giving access to the all of the function in the package. It helps interpreter to recognize the folder as package if it empty then this means that all the function from each module will be available or can be imported.

ML & AI

* Numpy
* Pandas
* SciPy

Packages for web development

* Flask
* Django
* Fast API

Deep Learning

* Scikit-Learn
* TensorFlow

Data Visualization

* Matplotlib
* Seaborn
* Plotly

Computer Vision

* OpenCV

GUI Application

* Tkinter
* PyQT5
* Kivy

Natural Language Processing

* NLTK

Tkinter

Tkinter is a standard python GUI Library that provides a set of tool and widgets to create desktop application with graphical interface.

Sieve of Eratostheses, Sieve of Eratostheses is a method for finding all primes upto a given natural number.

Concept, firstly we get all the values up to given natural no., then we select one number and removes its multiple, then select second no. and remove its multiple and so on. At the end we gets prime nos. only.

Code:

Method 1:

def prime(n):

list=[ ]

for i in range (2,n+1):

if i not in list:

print(i)

for j in range (i\*i,n+1,i):

list.append(j)

return list

p\_no = prime(30)

print(p\_no)

Method 2:

def prime(num):

l = [True for i in range(num+1)]

p = 2

while (p \* p <= num):

if (l[p] == True):

for i in range(p \* p, num+1, p):

l[i] = False

p += 1

for p in range(2, num+1):

if l[p]:

print(p)

prime(30)