Why Python demand in current Industry ?

* Python is in great demand. Its popularity is expanding by the day.
* It has the most **promising career opportunities**.
* It is also **easier to write** and **code** than any other programming language.
* Python is so well-built. It can quickly fulfill any demand. All you have to use is its **high- level libraries** like **, Matplotlib, NumPy, and Pandas** and **frameworks** like **Django** Framework**, Flask** Framework are some of them.
* That's one of the reasons it's the world's most popular programming language.
* Tech companies use **artificial intelligence (AI)** and **machine learning (ML)** systems. Python is compatible with both. This is thanks to the Python libraries such as NumPy and Pandas. Thus, the demand for Python developers is high.
* Python is the perfect choice for developing both complex and small-scale apps.

Here are a couple of more reasons why Python is in high demand.

* Python is an **open-source** programming language. This implies that anybody can use it. And that's for free. You can access it at any time and from any location.
* Python is one of the **simplest programming** languages. It is less difficult to learn than JavaScript and C++.
* Python is a **general-purpose** programming language. You can use it for anything. Complex apps or simple web design. It is not designed for one problem.
* Python is a **cross-platform** language. This means that one developer can create an app suitable for any device and platform.
* It is **fast** and **reliable**. This means that it has a faster time-to-market.
* Appropriate for **data science** and **analytics**. Python is simple to learn and adaptable. As a result, it is ideal for individuals who work with **machine learning**, **cloud computing**, and **big data**
* It has **custom automation.** Python offers tools that help you automate repetitive tasks.
* Python is cost-effective. These engineers earn high wages. Yet, they are less expensive than JavaScript developers.
* Compatible with **IoT Tech**. You can use Python with the Internet of Things technology.

### Installation of Python Software:

* + Download the required python latest software version from the following website

**https://**[**www.python.org/downloads/**](http://www.python.org/downloads/)

* + Select **windows** option and click **download button**
  + **Double click** on the downloaded Installer file and install it by following steps.
  + Latest Python 3 release is : **3.X.Y** version

**Adding Python PATH Environment Variables:**

After downloading and installing Python Software, by default it is installing in C-Drive likebelow,

**PATH** : **C:\Users\Lenovo\AppData\Local\Programs\Python\Python311**

* + It is required for **adding** Python **Interpreter** to the Environment variable section.

**PATH : C:\Users\Lenovo\AppData\Local\Programs\Python\Python311\Scripts**

* + It is required for **adding** Python **PIP** command to the Environment variable section.
  + Using PIP we can install Third party Python Packages/Modules.

**C:\Users\Lenovo> pip install required\_module\_Name**

**Development of Python application or Programs**

* + We can develop python applications or programs in 2 modes. They are,

1. Interactive mode
2. Batch Mode
3. **Interactive mode**
   * Interactive mode is a command line shell.
   * In command line shell if we write any Python statement immediatly that statement willexecute and gives the result.
   * Interactive mode is used to test the features of the Python ( What Python can do

?)

* + Open command prompt and enter

**python**.C:\Users\Lenovo> **python**

* + Now we are going to get python prompt area like below

**>>>**

* + Now we can test python coding basics like below

>>> a = 10

>>>

pri nt(

a) 10

>>> type(a)

<class 'int'>

>>>

print(“He llo”)Hello

* + NOTE: Interactive mode is not used for development of the business applications.
  + To overcome this problem then we can use Batch Mode.

**2 . Batch mode**

* + In Batch mode we write group of python statements in any one of the Editors or IDE's.

**Different Editors are :**

1. Notepad
2. Notepad++
3. EditPlus
4. nano
5. IDLE

**Different IDE's are :**

1. Pycharm
2. Visual Studio.
3. Sublimetext
4. Atom
5. Eclipse
6. NetBeens



* + After writng the group of Python statements in anyone of the Editor or IDE , we savethe Python file with extension **.py**

For example : FileName.py

* + After developing the .py files , we submit those files to the python

**Interpretor**directly.

* + Batch mode is used for developent of business applications.

For Example :

Step1: Open the Notepad editor Step2: Write the following code

i = 10

j = 20

print( i + j )

Step3: Save the above file with Basic.py in any one location.

Stem4: Open the command prompt upto python file storing location (Basic.py) and execute python file.

D:\Python\_Programs> **python Basic.py** >> click enter

Output :

300

100

**Note : IDLE --->> Integrated Development Language Environment**

* + When ever we install python software in windows os along with the python softwareIDLE or python GUI editor will be installed.
  + In IDLE editor we can develop the Python programs in interactive and batch mode.
  + After developing the python application in IDLE editor directly we can run that pythonby using shortcut key(F5).
  + Within the IDLE editor we can perform the debugging operation of the pythonapplications.

**How to Create a python program using IDLE tool ?**

Open **IDLE** tool --->> Click on **File** menu --->> Click on **New File Ctrl + N** >> Then

it will open a python Notepad prompt --->> Now create a python program.

Then Cleck on **File** menu >> Click on **Save Ctrl + S**

---->> Then it will open Python software location prompt >> Choose your required

folder

wherever you want to save ---->> Give file any file name and save with .py extension like **Sample.py**

**How to Execute a python program using IDLE tool ?**

Click on **Run** menu --->> Click on **Run Module F5 >>** Now output will show on

IDLE shell.

##### Advantages of Learning Python.

* The [**Python developer career**](https://teamcubate.com/blogs/the-python-developer-career)can be **very profitable**. Since the language is so popular, many companies are searching for such developers. And, the money they are willing to pay to those employees is more than great.
* **You can work various job types, including:**
  + Python web developer
  + Machine Learning engineer.
  + Artificial intelligence .
  + Software engineer engineer.
  + Data analyst
  + Product manager
  + Senior data scientist
  + Automation testing engineer
  + Cyber security expert
  + Database administrator
  + DevOps Engineer
  + Gaming/Animation developer
  + E-Commerce developer

Finally Continuous learning and skill development are essential. They will be able to keep up with the newest technologies this way.

**Generally Every website has 3 ends,**

1. Frontend part
2. Backend part
3. Database part
4. **Frontend part:**
   * A person who knows Frontend work very well and who can do any part of frontend work is called Frontend Developer.
   * Every Frontend Developer must know about all UI Technologies. For example, **HTML , CSS , Bootstrap , JavaScript**, etc…
   * A good Froentend developer can easily solve user problems and he/she must be responsible for frontend designing.
5. **Backend part:**
   * A person who knows Backend work very well and who can do any part of backend coding is called Backend Developer.
   * Every Backend Developer must know any one or two backend technologies like Python or Java or .Net or PHP.....
   * A good Backend Developer can easily write the logics which are required to handle the server or database and he/she must be responsible for any kind of backend work.
6. **Database part:**
   * A person who knows database logics and work on database side then call database developer.
   * DB developers required to learn at least any one database like MySQL, Mongo DB, etc..

**Full Stack Web Developer:**

* + A person who can do both frontend and backend and database works very well is called Full Stack Web Developer.
  + Stack means layer, so full stack means all layers.
  + A person who knows all layers of website is called Full Stack Web Developer.
  + Full Stack Web Developer must know about database also along with frontend and backend technologies.

**About UI Technologies:**

The Frontend Technologies are nothing but UI Technologies.

1. UI stands for User Interface.
2. An interface is a set of items or menus through which a user communicates with a Server.
3. UI is a junction between a user and a computer program.
4. UI Technologies are
   1. HTML
   2. CSS
   3. Bootstrap
   4. Javascript
   5. jQuery
   6. DOM
5. What is Program ?
   * A Set of instructions to perform perticular task is called as program.
6. What is Software ?
   * Set of programs to perform multiple task is called as Software.
7. what is the purpose of program or Software ?
   * To communicate with electronic device.
   * One electronic device to communicate with another electronic device.
   * To make things easy to reduce humans works we need program or software. For example : Bank Software
8. How to develop software ?

By using programming languages we can develop.

1. Why general purpose python is a programming language ?

Python is used to develope not only one type of application , it is used in more than one type of applications.

###### Applications of Python:

The following real time applications are developed by using PYTHON programming,

1. Web Applications
   1. Java ---> Servlets, JSP,..etc
   2. C#.Net ---> ASP.Net
   3. Python--- > Django, Flask, Pyramid, etc..
2. Artificial Inteligence (A.I) Applications
3. Machine learning Applications
4. Deep learning Applications
5. Data Science Applications
6. Devopps applications
7. Text Processing Applications
8. Image Processing Applications
9. Web Scrapping / Harvesting Applications.
10. Health care Applications
11. Testing applications
12. Games/Animation applications
13. IOT applications
14. Robotics applications
15. Data Visulization Applications
16. Embeded Systems
17. Languages Development
18. Automation Testing
19. Operating Systems
20. Console Based Applications
21. GUI based Applications.

About Python Indroduction :-

* Python is a Powerful , general purpose, dynamic, high-level, and interpreted programming language.
* It supports Object Oriented programming approach to develop applications.
* It is simple and easy to learn and provides lots of high-level data structures.
* Python is easy to learn , powerful and versatile scripting language, which makes it attractive for Application Development.

History of Python Programming Language :

* Python Programming Language concevied in the year 1980.
* Python Programming Language implementation begin in the year 1989.
* Python Programming Language officially released in the year 1991 Feb 21.
* Python Programming Language was developed by the "Guido Van Rossum".
* Guido Van Rossum was developed Python Programming Language at mathematical reaserch institute called as CWI (Centrum Wiskunde Informatica), which is located at "Netherlands".
* The Predecessor of Python Programming Language is ABC programming language.

**Python Programming language Inspired from What languages?**

Guido Van Rossum developed the python language by taking the different language features. They are like,

* 1. Functional Oriented Programming language like ---->> C-language
  2. Object Oriented Programming language like ---->> C++ , Java
  3. Scripting Oriented Programming language like --- >> PERL ,Shell Script
  4. Modular Oriented Programming language like ---->> Modulo3

[**Why is it called Python?**](https://docs.python.org/3/faq/general.html)

* When Guido Van Rossum began implementing Python, Guido van Rossum was also reading the published scripts from [“Monty Python’s Flying Circus”](https://en.wikipedia.org/wiki/Monty_Python), it is a BBC comedy series from the 1970s.
* Guido Van Rossum thought he needed a name that was short, unique, and slightly mysterious, so he decided to call the language **Python**.

Versions of Python:

* + Python Programming Language contains 3 types of Versions Majorly. They are,
    1. Python 1.X where 1 represents Major version and X represents minor versions like 0,1,2,3,4,5,6
    2. Python 2.X where 2 represents Major version and X represents minor versions like 0,1,2,3,4,5,6,7
  + Python 3.x where 3 represents Major version and X represents minor versions like 0,1,2,3,4,5,6,7,8,9,10,11,12
  + Python 3.X does not have backword compatability with Python 2.x
  + Python Software is officially maintained by a Non-Commericial Organization called "**Python Software Foundation(PSF)**".
  + The official web site for Python Software downloading is [**www.python.org/**](http://www.python.org/)

##### Features of Python Language.

* + Features of a language are nothing but Services / Facilities Provided by Language.
  + Developers and those features are used by Real Time programmers for developing real time applications.

For example : latest Mobiles features.

1. Simple and easy to learn.
2. Easy-to-read
3. Expressive Language
4. Free and Open Source
5. Dynamic Memory Allocation
6. Interpreted Language
7. Cross-platform or Portable Language
8. Object-Oriented Language
9. Databases
10. Embeddable
11. Extensible
12. Scalable
13. Automatic garbage collecter

##### Features of Python Explanation.

1. Simple and easy to learn:
   * Python is having simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
2. Easy-to-read:
   * Python code is more clearly defined and visible to the eyes.
3. Expressive Language :
   * Python can perform complex tasks using a few lines of code.
   * A simple example, the hello world program you simply type print("Hello World").
   * It will take only one line to execute, while Java or C takes multiple lines.
4. Free and Open Source :
   * Python is freely available for everyone.
   * It is freely available on its official website [www.python.org.](http://www.python.org/)
   * It has a large community across the world that is dedicatedly working towards make new python modules and functions.
   * Anyone can contribute to the Python community.
   * The open-source means, "**Anyone can download its source code without paying any money**".
5. Dynamic Memory Allocation:
   * In Python, we don't need to specify the data-type of the variable.
   * When we assign some value to the variable, it automatically allocates the memory to the variable at run time.
   * Suppose we are assigned integer value 15 to x, then we don't need to write int x = 15. Just write x = 15.
6. Interpreted Language:
   * Python is an interpreted language, it means the Python program is executed one line at a time.
   * The advantage of being interpreted language, it makes debugging easy and portable.
7. Cross-platform or Portable Language:
   * Python can run equally on different platforms such as Windows, Linux, UNIX, and Mac os, etc.
   * So, we can say that Python is a portable language.
   * It enables programmers to develop the software for several competing platforms by writing a program only once.
8. Object-Oriented programming Language:
   * Python supports object-oriented language and concepts of classes and objects come into existence.
   * It supports inheritance, polymorphism, and encapsulation, etc.
   * The object-oriented procedure helps to programmer to write reusable code and develop applications in less code.
9. Databases:
   * Python provides interfaces or modules to communicating with all major commercial databases.

For example, MySQL DB, Oracle DB, SQL Server DB , Mongo DB, etc...

* + Using python we can communicate with different databases by installing those interfaces or modules using PIP command.

For example : pymysql , mysqlclient, cx\_Oracle, mongodb, etc.... Syntax : **pip install pymysql**

1. Embeddable:
   * The code of the other programming language can use in the Python source code.
   * We can use Python source code in another programming language as well.
   * It can embed other language into our code.
2. Extensible:
   * You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
3. Scalable:
   * Python provides a better structure and support for large programs than shell scripting.
   * We can develop using python low level , mediam level and high level programs.
4. Automatic garbage collecter
   * Python supports automatic garbage collection. For example,

a = 10

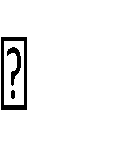
b = 20

a = 30

print(a)

Output: 30 # but not 10. It means ‘a’ variable value replaced by 30 value.

* + Here 10 value is initially assigned to a variable, next 20 assigned to b variable and 30 assigned again to same a variable only.
  + So 10 value is replaced by 30 in variable a .

So python garbage collector automatically removes a = 10 statement.

# What is Comments and Python ?

In Python, comments are used to annotate code with explanations, notes, or reminders.

Comments are ignored by the Python interpreter during program execution and Python supports two types of comments: They are single-line comments and multi-line comments.



1. Single-line comments:

Single-line comments begin with the hash character (**#**) and continue until the end of the line.

Anything after the hash character on that line is considered a comment.

For example:

>>> # This is a single-line comment

>>> print("Hello, World!") # This is also a comment

1. Multi-line comments:

Python does not not have built-in multi-line comment syntax like some other programming languages, such as C or Java.

However, you can achieve multi-line comments, in Python by enclosing multiple lines of text within triple quotes (**'''** or **"""**).

Although not technically comments, Python treats these triple-quoted strings as literals and ignores them unless they are assigned to a variable or used as docstrings (documentation strings).

For example:

'''

This is a multi-line comment. It spans across multiple lines. '''

"""

This is also a multi-line comment.

It can be enclosed in triple single quotes or triple double quotes. """

print("Hello, World!")

While triple-quoted strings can serve as a workaround for multi-line comments, they are generally used more commonly for docstrings to provide documentation for modules, classes, functions, or methods.



**Indentation Concept of Python**

In Python, indentation plays a crucial role in defining the structure and flow of the

code. Unlike many other programming languages that use braces or keywords

**end**

**begin**

**{}**

like

and

to denote blocks of code, Python uses indentation to indicate

blocks of code.

The key points about Python indentation are:

1. **Indentation Levels**: Python code blocks are defined by their indentation level. A block of code with the same level of indentation is considered part of the same block.
2. **Whitespace**: Python uses whitespace (spaces or tabs) to define indentation. While either spaces or tabs can be used for indentation, it's recommended to be consistent within your codebase. Most Python style guides, including PEP 8 (the official Python style guide), recommend using four spaces for each indentation level.
3. **Indentation Errors**: Incorrect indentation can lead to syntax errors or unintended behavior in Python programs. For example, if the indentation within a block is inconsistent, Python raises an **IndentationError**. It's essential to ensure that all lines within the same block have the same indentation level.
4. **No Explicit Block Delimiters**: Unlike many other programming languages, Python does not use explicit block delimiters (such as braces **{}**) to mark the beginning and end of blocks. Instead, the indentation level determines the structure of the code. This makes Python code more readable but also requires careful attention to indentation.

Example:

if True:

print("This line is indented, so it's part of the if block.") if False:

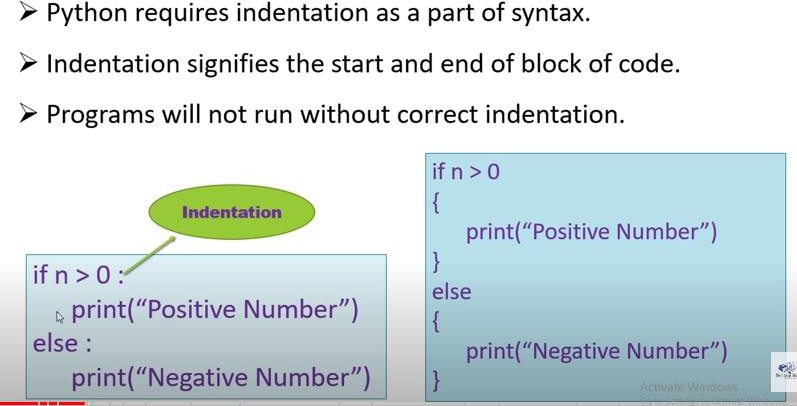
print("This line is further indented, so it's part of the nested if block.") print("This line is still part of the outer if block.")

print("This line is not indented, so it's outside of the if block.")

In the above example, the lines indented under the statements belong to those blocks. The statement outside of any indentation is not part of any block and is executed unconditionally.

**print**

**if**



**In Java**

**In Python**

Follow the below structure:

In summary, Python's indentation concept enhances code readability and enforces a consistent coding style. It's essential to pay attention to indentation while writing Python code to avoid syntax errors and maintain clarity.

**Use of Quotes and different types**

In Python, quotes are used to denote strings, which are sequences of characters. There are three types of quotes used for defining strings: single quotes (**'**), double quotes (**"**), and triple quotes (**'''** or **"""**). Each type has its specific use cases and behavior.

|  |  |  |
| --- | --- | --- |
| **Single Quotes (') and Double Quotes (")** | |  |
|  | * Single quotes and double quotes can be used interchangeably to define strings. * They are useful when you need to include apostrophes or quotation marks inside a string without escaping them. * ***Example:*** | |

single\_quoted\_string = 'This is a single-quoted string.' double\_quoted\_string = "This is a double-quoted string."

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Triple Quotes (''' or `""")** | | |  |
|  | | * Triple quotes allow you to create multiline strings in Python. * They preserve the line breaks and formatting within the string. * Triple single quotes (**'''**) and triple double quotes (**"""**) are functionally equivalent. * They are commonly used for docstrings (documentation strings) to provide documentation for functions, classes, modules, or scripts.   **Example:** | | |
|  | | | multiline\_string = '''This is a multiline  string.''' | |
|  | | * Triple quotes are also used when you need to include single quotes or double quotes inside a string without escaping them.   **Example:** | | |
|  | | | triple\_quoted\_string = '''She said, "Hello!"'''  Triple quotes are useful for defining regular expressions, SQL queries, or other text  blocks that span multiple lines. | |
|  | | In summary, single quotes and double quotes are primarily used for single-line strings, while triple quotes are used for multiline strings, docstrings, or strings containing both single and double quotes.  The choice between single quotes and double quotes is mostly a matter of personal preference or coding style conventions within a project. | | |
|  | | | | |

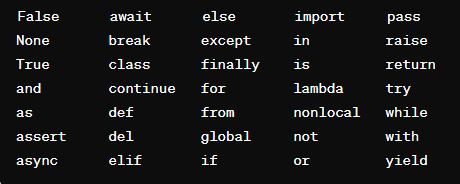
# What is python keywords?

Python keywords are reserved words that are predefined by the Python language and have special meanings and functionalities.

These keywords cannot be used as identifiers (such as variable names or function names) in your Python programs.



Python's keywords are case-sensitive and must be written exactly as shown. Here is a list of Python keywords as of Python 3.9:



In python we have totally 35 keywords. Out of 35 only 3 keywords first letters starts with upper case. They are,

False True None

You can use or importing predefined modules using **import** keyword.

You can also retrieve the list of keywords programmatically using the predefined module in Python:

**keyword**

>>> import keyword

>>> print( keyword.kwlist )

This will print the list of Python keywords as shown above.

### What is an Identifier ? Purpose of an Identifier ?

In programming, an identifier is a name given to entities such as **variables**, **functions**, **classes**, **modules**, or any other user-defined **objects**. Identifiers are used to uniquely identify these entities within a program.

**In Python, identifiers follow certain rules:**

1. **Character Set**: Identifiers can consist of letters (both lowercase and uppercase), digits, and underscores (**\_**). However, they cannot start with a digit.
2. **Case Sensitivity**: Python is case-sensitive, meaning **myVar**, **MyVar**, and considered different identifiers.

**myvar**

are all

1. **Reserved Keywords**: Identifiers cannot be the same as Python keywords (also known as reserved words), which have special meanings in the language. Examples of keywords include **if**, **else**, **for**, **while**, **def**, **class**, **import**, etc.
2. **Length**: There is no limit on the length of an identifier, but it's recommended to keep them reasonably short for readability.

**Convention**: While not enforced by the Python interpreter, there are conventions for naming identifiers to enhance code readability.

**For example:**

* + Use descriptive names that convey the purpose or meaning of the entity.
  + Use lowercase letters for variable names and function names, and separate words with underscores for readability (snake\_case).
  + Use uppercase letters for constants.
  + Use CamelCase for class names.
  + Avoid using single-character names except for loop variables (**i**, **j**, **k**).

**Examples of valid identifiers:**

my\_var variable\_1 some\_function PI

MyClass

**Examples of invalid identifiers:**

123var # Cannot start with a digit

if # Cannot use reserved keywords as identifiers some-var # Hyphens are not allowed

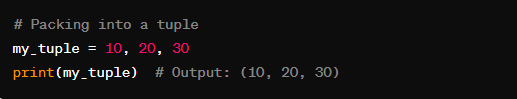
my\_var! # Special characters like '!' are not allowed



Following these rules ensures that your identifiers are valid and adhere to Python's naming conventions, leading to more readable and maintainable code.

##### Python Paking and Unpaking Concepts?

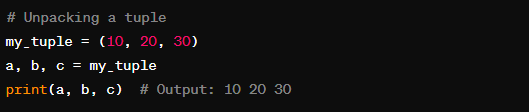
In Python, packing and unpacking refer to the operations of collecting multiple values into a single variable (packing) or spreading values from an iterable into individual variables (unpacking).



**Packing**

Packing is the process of combining multiple values into a single variable or data structure, typically a tuple. You can achieve packing by simply listing the values separated by commas:

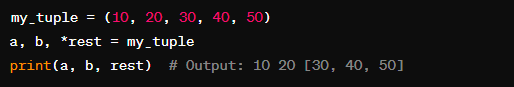
You can also explicitly use parentheses for clarity:



**Unpacking**

Unpacking involves extracting individual elements from a data structure like a tuple or a list and assigning them to separate variables.

Unpacking can also be used to unpack only a part of the sequence:



Here, will contain the remaining elements as a list.

**Packing and Unpacking with Functions**

Packing and unpacking are often used in functions. For example, you can use the number of arguments in a function definition:

**\***

**rest**

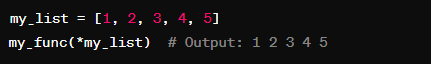
operator to accept any



**\*\***

Similarly, you can use packing to pass multiple arguments to a function:

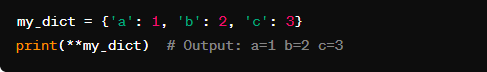
**\*my\_list**



In this case, unpacks the list into individual arguments.

**Packing and Unpacking with Dictionaries**

You can also use packing and unpacking with dictionaries using operator:



This is used mainly for passing keyword arguments to functions.

Understanding packing and unpacking is crucial for writing concise and readable Python code, especially when dealing with functions that accept variable numbers of arguments or returning multiple values from functions.

**Python comments, identifiers, variables, and keywords:**

**Comments:**

**What are comments in Python? Why are they used?**

**What is the purpose of comments in code? Can you give some examples?**

**How do you write single-line comments in Python?**

**How do you write multi-line comments in Python?**

**Explain the different types of comments supported in Python.**

1.

2.

3.

4.

5.

**Identifiers:**

**Are identifiers case-sensitive in Python?**

**Give examples of valid and invalid identifiers in Python.**

**Can identifiers start with an underscore symbol in Python?**

**Can identifiers start with a letter in Python?**

**Can identifiers start with a digit in Python?**

**What are the rules for naming identifiers in Python?**

**What are identifiers in Python?**

**What is identifier in Python? What is purpose of identifier?**

1.

2.

3.

4.

5.

6.

7.

8.

**Variables:**

**Can you change the data type of a variable after it has been declared in Python?**

**What are the possible ways to assign values to variables?**

**What is variable naming convention in Python?**

**Explain the concept of variable scope in Python.**

**How are variables declared and assigned values in Python?**

**What are variables in Python?**

**What is the purpose of variable?**

**What is variable in Python?**

1.

2.

3.

4.

5.

6.

7.

8.

**Keywords:**

**Explain the significance of the global and nonlocal keywords in Python.**

**How do you check if a given word is a keyword in Python?**

**Why can't we use keywords as identifiers in Python?**

**How many keywords are there in Python? Can you name a few?**

**What is purpose of the keyword in python?**

**What are keywords in Python?**

1.

2.

3.

4.

5.

6.

### What is Operator in python and Types of operators

In Python, operators are special symbols or keywords that perform operations on operands (variables or values). Python supports various types of operators, which can be classified into several categories:

**We have several types of operators in python. They are,**

* Arithmetic Operators
* Relational / Comparison Operators
* Logical Operators
* Assignment Operators
* Membership Operators
* Identity Operators
* Ternary operator

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Arithmetic Operators:**  Arithmetic operators perform mathematical operations like addition, subtraction, multiplication, division, etc.  Arithmetic operators returns **Number** type result like **int, float** | | |
| * Addition: | | **+** |  |
| * Subtraction: **-** * Multiplication: **\*** * Division: **/** * Modulus (Remainder): **%** * Exponentiation: **\*\*** * Floor Division (Integer Division): **//** | | | |
|  | **For Example:** |  | |

a = 10

b = 3

print(a + b) *# Output: 13*

print(a - b) *# Output: 7*

print(a \* b) *# Output: 30*

print(a / b) *# Output: 3.3333333333333335*

print(a % b) *# Output: 1* print(a \*\* b) *# Output: 1000* print(a // b) *# Output: 3*

Comparison Operators

|  |  |  |  |
| --- | --- | --- | --- |
| * Equal to: **==** * Not equal to: **!=** * Greater than: **>** * Less than: **<** * Greater than or equal to: **>=** | | | |
| * Less than or equal to: | | **<=** |  |
|  |  | | |
| **For example:**  x = 5  y = 10  print(x == y) *# Output: False* print(x != y) *# Output: True* print(x > y) *# Output: False*  print(x < y) *# Output: True*  print(x >= y) *# Output: False*  print(x <= y) *# Output: True* | | | |

**Logical Operators:**

Logical operators are used to combine conditional statements. Returns **True** or **False** values.

|  |  |  |  |
| --- | --- | --- | --- |
| * Logical AND: | | **and** |  |
| * Logical OR: | **or** |  | |
| * Logical NOT: **not**   **and truth table**  **condition1 condition2 Result**  True True True  True False False  False True False  False False False  **or truth table**  **condition1 condition2 Result**  True True True  True False True  False True True  False False False | | | |



**Note: not** operator returns **True** value as **False** and **False** value as **True**

**For example**

a = True

b = False

print(a and b) *# Output: False*

print(a or b) *# Output: True*

print(not a) *# Output: False*

**Assignment Operators:**

Assignment operators are used to assign values to variables.

* Assignment:
* Add and assign: **+=**
* Subtract and assign: **-=**
* Multiply and assign: **\*=**
* Divide and assign: **/=**
* Modulus and assign: **%=**
* Exponentiation and assign:
* Floor division and assign: **//=**

*x = x + 2*

*# Equivalent to: # Output: 7*

**For Example:**

x = 5

x += 2

print(x)

**\*\*=**

**=**

**Membership Operators:**

Membership operators are used to check if a value or variable is found in a sequence.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **in** | : Returns True if a value is found in the specified sequence. | |
| **not in** | | : Returns True if a value is not found in the specified sequence. |
| **For example:** | | | |

my\_list = [1, 2, 3, 4, 5]

print(3 in my\_list) *# Output: True*

print(6 not in my\_list) *# Output: True*

|  |  |  |
| --- | --- | --- |
|  | **print** |  |
| **x** | and | **y** |



**Identity Operators:**

Identity operators are used to compare the memory locations of two objects.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **is** | : Returns True if both variables point to the same object. | |
| **is not** | | : Returns True if both variables point to different objects. |
| **For example:** | | | |

x = [1, 2, 3]

y = [1, 2, 3]

z = x

print(x is z) *# Output: True* print(x is y) *# Output: False* print(x is not y) *# Output: True*

**Ternary Operator:**

Python supports the ternary operator, also known as the conditional expression. The syntax of the ternary operator in Python is:

**x if condition else y**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| This expression evaluates to | **x** | if the condition is true, and | **y** | otherwise. |
| **Here's an example demonstrating the use of the ternary operator in Python:**  *# Assigning a value based on a condition*  x = 10  y = 20  result = x if x > y else y print(result) *# Output: 20*  *# Using the ternary operator in a print statement*  print("x is greater than y" if x > y else "y is greater than or equal to x")  *# Output: "y is greater than or equal to x"*  In the first example, the value of **result** is assigned **x** if **x** is greater than **y**, otherwise, it's assigned **y**.  In the second example, the ternary operator is used directly within the statement to determine which message to print based on the comparison between . | | | | |

|  |  |  |
| --- | --- | --- |
|  | |  |
|  | The ternary operator provides a concise and readable way to express conditional expressions in Python. | |
| These are the main types of operators in Python, each serving different purposes and used in various contexts within Python programming. Understanding them is crucial for effective programming and problem-solving. | | |



|  |  |  |  |
| --- | --- | --- | --- |
| **Fundamental Data Types** | **vs** | **Collection Data Types** | **:** |
| In programming, data types classify different types of data that can be used in a program. They define the operations that can be performed on the data and the way the data is stored in memory.  Two broad categories of data types are: | | | |

1. **Fundamental Data Types**:
   * Fundamental data types represent single values.
   * They are basic building blocks and include primitive data types such as integers, floating-point numbers, characters, and boolean values.
   * In many programming languages, fundamental data types are also called primitive data types.

**Here are some examples:**

1. **Integer:** Represents whole numbers (e.g., 1, 10, -5).
2. **Float:** Represents floating-point numbers with decimal values (e.g., 3.14, -0.5, 1e3).
3. **Character:** Represents single characters (e.g., 'a', 'X', '@').
4. **Strings :** Represents group or collection of characters (e.g., 'python' , 'JSRao' )
5. **Boolean:** Represents true or false values.

Fundamental data types are typically have fixed memory allocations and are immutable, meaning their values cannot be changed after they are created.

1. **Collection Data Types**:
   * Collection data types, also known as composite data types or aggregate data types, represent collections or groups of values.
   * They allow you to store multiple values in a single variable and are used to manage and manipulate collections of data more efficiently.

**Collection data types include:**



|  |  |  |
| --- | --- | --- |
|  | * **Lists:** Ordered collections of items, where each item can be of any data type. Lists are mutable, meaning you can change their elements after they are created. * **Tuples:** Similar to lists but immutable, meaning their elements cannot be changed after they are created. * **Sets:** Unordered collections of unique items. Sets do not allow duplicate elements. * **Dictionaries:** Key-value pairs where each value is associated with a unique key. Dictionaries are mutable and allow you to access, modify, and delete items by their keys. | |
|  | |  |
| Collection data types provide various methods and operations for manipulating and accessing their elements, such as indexing, slicing, appending, and removing. | | |

**In summary, fundamental data types represent single values with fixed memory allocations, while collection data types represent collections or groups of values and provide more complex data structures and operations for managing and manipulating collections of data.**

###### How to read the data from keyboard using python.

To read data from the keyboard (standard input or stdin) in Python, you can use the function. It reads a line of text from the standard input (stdin) and returns it as a string.

**input()**

**Example 1: Simple Input and Output** *# Prompting the user to enter their name* name = input("Enter your name: ")

*# Printing a greeting message using the input provided by the user*

print("Hello, " + name + "! Welcome to Python programming.")

In this example, the function prompts the user to enter their name. The entered name

is stored in the variable **name**, and then a greeting message is printed using that name.

**input()**

**Example 2: Simple Input and Output**

data = input("Enter some data: ") print("You entered:", data)

When you run this script, it will prompt you to enter some data. Once you've entered the data

and pressed Enter, it will be stored in the variable **data**, and then it will print out what you entered.

Keep in mind that the

function always returns a string. If you need to convert the input

into a different type (like integer or float), you can use conversion functions like or

accordingly.

**For example:**

**float()**

**int()**

**input()**

number = int(input("Enter an integer: ")) # 20.50

This will read an integer from the user's input. If the input cannot be converted into an integer,

it will raise a exception. So, it's a good practice to handle exceptions when

**ValueError**

converting input data to different types.

**Example 2: Converting Input to Integer**

*# Prompting the user to enter an integer*

num = int(input("Enter an integer: "))

*# Doubling the entered integer and printing the result*

result = num \* 2

print("Double of", num, "is:", result)

In this example, the function is used to receive an integer input from the user. The

**int()**

**input()**

function is then used to convert the input (which is a string) to an integer so that arithmetic operations can be performed.

**Example 3: Handling Floating Point Input**

*# Prompting the user to enter a floating-point number*

value = float(input("Enter a floating-point number: "))

*# Squaring the entered number and printing the result*

squared\_value = value \*\* 2

print("Square of", value, "is:", squared\_value)

In this example, the function is used to receive a floating-point number input from the

**input()**

user. The function is then used to convert the input (which is a string) to a floating-

**float()**

point number for further computation.

**Example 4: Accepting Multiple Inputs**

*# Prompting the user to enter multiple values separated by commas*

numbers = input("Enter multiple numbers separated by commas: ")

*# Splitting the input string into a list of numbers*

number\_list = numbers.split(",")

*# Converting each number in the list to an integer*

integer\_list = [int(num) for num in number\_list]

*# Finding the sum of the entered numbers and printing the result*

total = sum(integer\_list)

print("Sum of the entered numbers:", total)

In this example, the **input()** function is used to receive multiple numbers separated by

commas from the user. The method is then used to split the input string into a list of numbers. Each number in the list is converted to an integer using list comprehension. Finally, the sum of the entered numbers is calculated and printed.

**These examples demonstrate different use cases of the**

**interaction with users through the command line interface.**

**function in Python, allowing**

**input()**

**split()**

#### Generally any organization contains 2 parts.

1. Data part
2. Operations part.

For example , Bank contains two parts .

* + In bank , Bank have Customers and Employees. Bank maintaining each person related data and providing the some services to each person.

For example,

1. **Customer**
   1. data ------>> c.name, c.address,cbal,…….
   2. operations >> deposit(),withdraw(),bal.eng()……
2. Employee
   1. data ----->> e.name, e.address,e.bal,…….
   2. operations ------>> da(),hra(),pf(),tax(),tsal().

Data :

* + Data of the any organisation we can represent by using any program language.
  + To represent the data of organisations all programming languages are providing **datatypes** and **variables**.
  + Every programming language supports datatypes and variables but the datatypes and variables of one language are not going to be same with another programming language.

Operations:

* + Operations of the organisations we can represent by using any programming language.
  + To represent the operations of the organisations every programming language is providing **functions** or **methods** or both.
  + Every programming language supports functions or methods but the functions or methods of one language are not going to be same with another programming language.

Data types:

* + Data types are nothing but some of the keywords of the programming languages, which are used to specify what type of data has to be store into the variables.
  + Without data types we cannot store the data into the variables.
  + Python supports the dynamic data types i.e, at the time of execution of program data type of the variable will be decided based on the data which is assigned to that variable.
  + for example, a = 10 >> here a is maintaining integer value 10.
  + At the time of writing the program, programmers should not specify data type to the variables explicitly otherwise we will get error.
  + for example, int a = 10 >> it returns error because of datatype int.

Python data types are categorized into two types.

1. Fundamental data types.
2. Collections data types.

1. Fundamental data types.

* The variables which are represented with fundamental data types are stores the address of the object in which we can represent only one element.
* Python supports the following fundamental data types

1. Numbers
   1. int ------->> a = 10
   2. float ------->> b = 10.50
   3. complex ------->> c = 2 + 3j
   4. bool ------->> d = True / False
2. Strings

a. str ------>> name = “Virat”

2) Collection data Types.

* The objects which are maintaining the group of elements at a time are called collection data types.
* Python is providing somany collection datatypes.

For example,

* 1. list type

lst = [10,20,30]

* 1. tuple type

tup = (10,20,30)

* 1. set type

set1 = {10,20,30}

* 1. dict type

dict1 = {1:10, 2:20, “name”:”Virat”, “age”:30}

For example,

l = [1,2,3,4]

##### Immutable and Mutable Objects:

* Everything in Python is an object.
* All objects in Python can be either mutable or immutable.
* So here , above all data types are again devided into 2 types.

1. ImMutable
   1. Numbers ( int , float , bool , complex )
   2. Strings ( str )
   3. Tuple ( tuple )
2. Mutable
   1. List ( list )
   2. Set ( set )
   3. Dictionary ( dict )

**Q. What is difference between Mutable and Immutable objects?**

* **A mutable object can be changed after it is created, and an immutable object can not be changed after it is created**.

**Built-In Functions:**

* Python supports the so many builds in functions.

1. type():
   * type() function is used to know the data types of variables.

For example,

>>> a = 10

>>> type(a)

<class 'int'>

>>>

1. id() :
   * id() function is used to know the address of the object which is pointed by the variables.
   * The built-in function id() returns the identity of an object as an integer.

for example,

>>> a = 10

>>> id(a) 1648190784

>>>

1. print():
   * print() function is used to display the data on console.

For example,

>>> a = 10

>>> print(a) 10

>>> print('Hello') Hello

##### Identifiying the differnet data types operations using + symbol :

1. int type + int type ------>> int type
   * Adding two integer type values will return result as Integer type only.
   * Here **+ operator** will perform **Arthematic** operation For example:

>>> 10 + 20 # 30

1. float type + float type ------>> float type
   * Adding two flaot type values will return result as Float type only.
   * Here **+ operator** will perform **Arthematic** operation For example:

>>> 10.0 + 20.5 # 30.5

1. int type + float type ------>> float type
   * Adding one integer type value and one float type value then it will return result as Float type only.
   * Here **+ operator** will perform **Arthematic** operation For example:

>>> 10 + 20.5 # 30.5

1. str type + str type ------>> str type
   * Adding two string type values will return result as String type only.
   * Here **+ operator** will perform **Concatination** operation. For example:

>>> “Hello” + “Python” # “HelloPython”

1. int type + str type ------>> TypeError
   * Adding one integer type value and one string type value then it will return result as exception like **TypeError**.
   * Here **+ operator** will perform **Arthematic** operation

For example:

>>> 10 + “Python” # TypeError

**TypeError**: unsupported operand type(s) for +: 'int' and 'str'

### Reading data from Keyboard:

* + Input to the program can come in various ways, for example from a database, from another computer, from mouse clicks and movements or from the internet.
  + Generally in most cases the input comes from the keyboard. For this purpose, Python provides the function input().
  + The input of the user will be returned as a string without any changes.
  + If this raw input has to be transformed into another data type needed by the algorithm, we can use either "casting (type casting)" functions or the "eval()".

For example :

* Direct input from user (default data type is string)

>>> n = input('Enter Number: ') # Enter Number: 10

>>> print(n) # 10

>>> type(n) # <class 'str'>

* Converting str type data by using int casting function

>>> n = int(input('Enter Number: ')) # Enter Number: 10

>>> print(n) # 10

>>> type(n) # <class 'int'>

* Converting str type of data by using "eval()" functions

>>> n1=eval(input('Enter Number: ')) # Enter Number: 10

>>> print(n1) # 10

>>> type(n1) # <class 'int'>

For example :

In interactive mode:

>>> name = input('What Is Your Name: ') What Is Your Name: Srinivas

>>> loc = input("What is Your Location: ") What is Your Location: Guntur

>>> print(" Hello " + name +", How are You. How is your " +loc+ ".");

Hello Srinivas , How are You. How is your Guntur.

In script mode:

* Open file and write the following code and save the file as fileName.py name = input("What is your Name : ")

loc = input("What is your Location : ")

print('Hello ' + name +', How are you. How is Your ' +loc+'.')

* If we run the file, we can see the following in the python screen. Output:

What is your Name : Srinivas What is your Location : Guntur

Hello Srinivas , How are you. How is Your Guntur.

Type Conversion Functions :

---- >> We can convert datatype in two different ways:

1. type casting functions
2. **eval()** function

Type casting functions

- >> These type conversion functions are used to convert string type data into

required types

1. int():

This int() function is used to convert into int data type.

For example : get two integers from the user and perform addition on those user values

Way-1

>>> a = input("Enter First Number: ") # Enter First Number: 10

>>> b = input("Enter Second Number: ") # Enter Second Number: 20

>>> print (a) # 10

>>> print (b) # 20

>>> type(a) # <class 'str'>

>>> type(b) # <class 'str'>

>>> id(a) # 52351712

>>> id(b) # 6925376

>>> c=a+b # adding two str variables

>>> print(c) # 1020

>>> type(c) # <class 'str'>

>>> id(c) # 52351136

>>> x=int(a) # converting str ‘a’ into int ‘x’

>>> y=int(b) # converting str ‘b’ into int ‘y’

>>> print(x) # 10

>>> print(y) # 20

>>> type(x) # <class 'int'>

>>> type(y) # <class 'int'>

>>> id(x) # 1625909520

>>> id(y) # 1625909680

>>> z=x+y # adding two int variables

>>> print(z) # 30

>>> type(z) # <class 'int'>

>>> id(z) # 1625909840

Way-2 (shortest way)

>>>print("The sum is " + str(int(input("Enter First Number: ")) + int(input("Enter Second Number: "))))

>>>Enter First Number: 10

>>>Enter Second Number: 20

>>>The sum is 30

1. Float():

This float() conversion function is used to convert other types into float type.

Way-1

>>> a=input("Enter First Number: ") Enter First Number: 10.5

>>> b=input("Enter Second Number") Enter Second Number20.5

>>> print (a) 10.5

>>> print(b) 20.5

>>> type(a) <class 'str'>

>>> type(b) <class 'str'>

>>> id(a) 52354240

>>> id(b) 52354144

>>> c=a+b

>>> print(c) 10.520.5

>>> type(c) <class 'str'>

>>> id(c) 52367096

>>> x=float(a)

>>> y=float(b)

>>> print(x) 10.5

>>> print(y) 20.5

>>> type(x) <class 'float'>

>>> type(y) <class 'float'>

>>> id(x) 47459712

>>> id(y) 47460096

>>> z=x+y

>>> print(z) 31.0

>>> type(z) <class 'float'>

>>> id(z) 46410368

Way-2 (shortest way)

>>>print(float(input("Enter First Number: ")) + float(input("Enter Second Number: ")))

>>>Enter First Number: 10.5

>>>Enter Second Number: 20.5

>>>31.0

1. Complex():

This complex() conversion function is used to convert string type data into Complex type.

For example :

Way-1

>>> a=input("Enter Number: ") Enter Number: 2+3j

>>> print(a) 2+3j

>>> type(a) <class 'str'>

>>> id(a) 52351712

>>> x=complex(a)

>>> print(x) (2+3j)

>>> type(x) <class 'complex'>

>>> id(x) 47539768

Way-2

>>>print(complex(input("Enter Number: ")))

>>>Enter Number: 4+6j

>>>(4+6j)

1. bool():
   * This bool() conversion function is used to convert string type data into Boolean type.

Way-1

>>>a=input("Enter either True or False: ") Enter either True or False: True

>>> print(a) True

>>> type(a) <class 'str'>

>>> id(a) 52354272

>>> x=bool(a)

>>> print(x) True

>>> type(x) <class 'bool'>

>>> id(x) 1625727600

Way-2

>>>print(bool(input("Enter Either True or False: ")))

>>>Enter Either True or False: True

>>>True

**eval()** function:

* + The python eval() function parses the expression passed to it and runs python expression(code) within the program.
  + eval() can identify the user input data type when user enterd data using keybord and stored in a variables.

Examples:

* + - Converting int value:

>>> a = eval(input('enter value for a: ')) enter value for a: 10

>>> print(a) # 10

>>> type(a) # <class 'int'>

* + - Converting float value

>>> b=eval(input('enter value for b: ')) enter value for b: 10.6

>>> print(b) # 10.6

>>> type(b) # <class 'float'>

* + - Converting string value

>>> c=eval(input("enter value for c: ")) enter value for c: 'JS Rao'

>>> print(c) # JS Rao

>>> type(c) # <class 'str'>

* + - Converting complex value

>>> d=eval(input('enter value for d: ')) enter value for d: 2+5j

>>> print(d) # (2+5j)

>>> type(d) # <class 'complex'>

* + - Converting bool value

>>> e=eval(input('enter value for e:'))

enter value for e:True

>>> print(e) # True

>>> type(e) # <class 'bool'>

#### Python Control Statements:

* + - Control statements are used to control the flow of program execution. These are classfied into three types.

1. Conditional or Decision Making or Selection Statements.
2. Looping or Iterative Statements.
3. Transfer or Jump Statements.
4. CONDITIONAL STATEMENTS:
   * Conditional statements will decide the execution of a block of code based on the expression.
   * The conditional statements return either True or False.
   * Python supports four types of conditional statements,
5. Simple IF or IF statement
6. IF – ELSE Statement
7. IF ELSE IF (elif) Statement
8. Nested if statement
9. if Statement
   * The Python if statement is same as it is with other programming languages.
   * It executes a set of statements conditionally, based on the value of a logical expression.
   * Here is the general form of one way if statement. Syntax:

if expression:

statement\_1 statement\_2

............

…………

* + In the above syntax, expression specifies the conditions it produces either true or false.
  + If the expression evaluates true then the same amount of indented statement(s) following it will be executed.
  + This group of the statement(s) is called a block.

Example 1: How to check a Student is passed or failed based on marks ?

>>> marks = int(input("Enter your marks to know pass or failed: "))

>>> if marks >= 35:

print("You are passed ...")

Output:

Enter your marks to know pass or failed: 45 You are passed ...

Example 2: Compare values and display message ?

a = int(input("Enter First Value: "))

b = int(input("Enter Second Value: "))

c = int(input("Enter third Value: ")) if a > b < c:

print("Hello")

Q ) How to test which is maximum value in two values ?

a = int(input("Enter First Value: "))

b = int(input("Enter Second Value: ")) max = a if (a > b) else b; print('Maximum value is :', max)

or

if a > b: x = a

else:

x = b

print('Maximum value is :', x)

1. if -- else Statement:
   * In Python if-else statement has two blocks, first block follows the expression and the other block follows the else clause. Here is the syntax.

Syntax:

if expression :

statement\_1 statement\_2

................

else :

statement\_3 statement\_4

.................

* + In the above case, if the expression evaluates to true then the same amount of indented statements(s) executed.
  + if the expression evaluates to false then same amount of indented statements(s) follow the else block will executed.

Example1: Write python script to check whether the given number is even or odd number ?

a = int(input('Enter Your number: ')) if a % 2 == 0:

print(a," is a even number") else:

print(a," is a odd number") Output:

Enter Your number: 20 20 is a even number Or

Enter Your number: 21 21 is a odd number

Example 2: Write a python script to know student is passed or failed ? marks = int(input("Enter your marks to know pass or failed: "))

if marks >= 35:

print("Your are passed") else:

print("Your are failed") Output:

Enter your marks to know pass or failed: 20

Your are failed` Or

Enter your marks to know pass or failed: 40 Your are passed

1. if else-if statement ( elif ) :

Example1: How to check a Student result based on the marks ? marks = int(input("Enter your percentage of marks : "))

if marks < 35 and marks > 0: print("You are failed")

elif marks >= 35 and marks < 50: print("Your got 3rd class")

elif marks >= 50 and marks < 60: print("You got 2nd class")

elif marks >= 60 and marks < 72: print("you got 1st class")

else:

print("you got distinction") Output:

Enter your percentage of marks : 50 You got 2nd class

Enter your percentage of marks : 90 you got distinction

Enter your percentage of marks : 20 You are failed

Example 2: Write a program to check food timings for eating ? time = int(input("Enter your time: "))

if time > 7 and time < 10:

print("Its time to have Breakfast..")

elif time >= 10 and time < 12: print("Its time to have Brunch..")

elif time >= 12 and time < 15: print("Its time to have Lunch..")

elif time >= 15 and time < 18: print("Its time to have Snacks")

elif time >= 18 and time < 20: print("Its time to have Dinner")

elif time >= 20 and time <= 24: print("Its sleeping time")

elif time >= 1 and time <= 7: print("Its sleeping time")

else:

print('you entered invalid time')

Output:

Enter your time: 8

Its time to have Breakfast.. Enter your time: 13

Its time to have Lunch..

Enter your time: -2

you entered invalid time Enter your time: 3

Its sleeping time

Example 3: How to find the biggest value of three given values ?

a = int(input("Enter First Value: "))

b = int(input("Enter Second Value: "))

c = int(input("Enter third Value: ")) if a > b and a > c:

print(a,'is greater then ',b, 'and',c) elif b > c:

print(b,'is greater then ',a, 'and',c) else:

print(c,'is greater then ',a,'and',b)

Output 1:

Enter First Value: 10 Enter Second Value: 20 Enter third Value: 15

20 is greater then 10 and 15

Output 2:

Enter First Value: 10 Enter Second Value: 5 Enter third Value: 30

30 is greater then 10 and 5

Output 3:

Enter First Value: 10 Enter Second Value: 3 Enter third Value: 5

10 is greater then 3 and 5

1. Nested if statement:
   * If we creating **if** statements inside another if statements , then we called as

**nested if** statments.

Example 1:

Database contains only male records, if user enters about female then display ‘female records are not available' and if user enters about male then check employees name, if male employee name existed then display his details, if that name is not available then display ‘nobody is there with that name’. if he enters

**any wrong gender then display ’you entered wrong gender’. Finally display ‘thank you’ at the end of result.**

Q ) male records displaying ? gender = input("Enter gender: ") name = input("Enter name : ")

if gender == "Female" or gender == "Male": if gender == 'Female':

print("Female records are not available") else:

if name == 'Satya' and gender == "Male": print("Satya is from hyd and working as SE")

elif name == 'Srinivas' and gender == "Male": print("Srinivas is from nagpura and working as ASE")

elif name == 'Nani' and gender == "Male": print("Nani is from Hyd and working as TL")

else:

print("No one is there with that name")

else:

print("You entered wrong gender: ") print("Thank")

Output:

Enter gender : Male Enter name : Satya

Satya is from hyd and working as SE Thank

Example 4: Male records display example gender=input("Enter your gender: ") name=input("Enter your name: ")

if gender == "Male" or gender == "Female": if gender != "Female":

if name == "Satya":

print("Satya is from Hyd and havaing 10 years exp") elif name == "Sai":

print("Sai is from mumbai and having 20 years exp") elif name == "Srinivas":

print("Srinivas is having 6 years exp and he is from hyd") else:

print("sorry, your name is not avaiable in the database")

else:

print("Sorry Female records are not available in the database..")

else:

print("Sorry, You entered invalid gender" ) print("Thank You ")

Output:

Enter your gender: Male1 Enter your name: Satya

Sorry, You entered invalid gender Thank You ......

Enter your gender: Male Enter your name: Sai

Sai is from mumbai and having 20 years exp Thank You ......

Enter your gender: Female Enter your name: Renu

Sorry Female records are not available in the database.. Thank You ......

Example2: Write a program to checking the eligibility of interview ? name = input("Enter your name: ")

qualification = input("Enter your qualification: ")

passedoutyear = eval(input("Enter your degree finished year: ")) percentage = eval(input("Enter your degree percentage: "))

if qualification == 'b.tech' or qualification == 'b.e':

if passedoutyear==2017 or passedoutyear == 2016: if percentage >= 60 and percentage <= 100:

print("Hey",name,"Congrates you are eligible for attending interview this week ")

elif percentage >= 45 and percentage < 60:

print("Hey",name,"You can attend the interview after two weeks") elif percentage >= 35 and percentage < 45:

print("Hey",name,"First you work in someother organizations and after two years you can try")

elif percentage < 35:

print("Hey",name,"Please dont try to attend interview in our organization ")

else:

print("Hey",name,"Please enter valid percentage") elif passedoutyear > 2017:

print("Hey",name,"you entered invalid year") else:

print("Hey",name,"You are not fresher")

else:

print("Hey",name,"Sorry, only btech or be students are eligible") print(qualification,"students are not eligible")

Q ) Write a program if student enters marks < 35 then display “You are failed”, if student enters >=35 and <=100 then display “You are passed”, if marks more then 100 then display “Enter proper marks” ?

Q. Write a program to check given number is Even or Odd number ?

Q. Write a program to check given number is Positive or Negative number ?

Q. Write a program to check wether the given Person is eligible for Vote or not ?

Q. Write a program to check beggest number among the given two numbers ?

Q. Write a program to check beggest number among the given Three numbers ?

Q. Write a program to check given number is divisible by 5 or not ?

Q. what is the use of range()?

---->> range() method is used to generate a sequence of numbers.

--->> range() method is accepting 3 parameters. syntax : range(start\_value, stope\_value, step\_value)

---->> default value for start\_value is 0

---->> default value for step\_value is 1

---->> stope\_value is must be user-defined value.stop\_value is always n-1.

For example,

range(10) -------->> range(0,10,1) >> 0,1,2,3,4,5,6,7,8,9

range(5,10)------->> range(5,10,1) >> 5,6,7,8,9

range(3,10,2)----->> range(3,10,2) >> 3,5,7,9

range(10,5,-1)---->> range(10,5,-1) >> 10,9,8,7,6

Example:

>>> for i in range(10): print(i)

output 0

1

2

3

4

5

6

7

8

9

>>>

---->> to print output in same line (horizontal) format then use end=' ' attribute inside print() method.

>>> for i in range(10):

print(i,end=' ') output

0 1 2 3 4 5 6 7 8 9

**Loops or Iterative Statements**

If you want to execute the certain statements multiple times then we have to use looping statements.

for loops :-

For loop is a programming language statement, i.e. an iteration statement, which allows a code block to be repeated a certain number of times.

**syntax : for item in sequence\_object :**

**print(item)**

styntax : for variable in sequence:

statement1 statement2 statement3

**For examples:**

st = "Python Developer" for i in st:

print(i)

lst = [1,2,3.5,"Python",4+5j,True] for i in lst:

print(i)

tup = (1,2,3,True,False,"Srinivas") for i in tup:

print(i)

se = {2,3,'Python',0,0,True,2+6j,'Srinivas'} for I in se:

print(i)

dic = {1:'a',2:'b','c':45}

for i in dic:

print(i)

**Q. Write a program to print the 10th table ?**

for I in range(1,11): r = 10 \* i

print(10 , '\*', i, '=', r)

**Output**

10 \* 1 = 10

10 \* 2 = 20

10 \* 3 = 30

10 \* 4 = 40

10 \* 5 = 50

10 \* 6 = 60

10 \* 7 = 70

10 \* 8 = 80

10 \* 9 = 90

10 \* 10 = 100

>>>

Q. WAP to for-loop to generate 10 to 1 numbers ? for i in range(10,0,-1):

print(i, end=' ') print('Thank you')

Q. WAP to display 1 to n even numbers?

Q. WAP to display sum of 1 to n numbers by using for-loop ? n = int(input('Enter any number :'))

sum = 0

for i in range(1, n+1): sum = sum+i

print(sum)

output

Enter any number :4 10

**While - loop**

* while loop is used to execute certain statements multiple times.
* in while loop first it checks the condition, if it is true then it will execute the statements. This process is continueous until while loop condition becomes false.

**Syntax:**

**while (condition): statement1 statement2 statement3 statement4**

**Example:**

num = 1

while(num <= 5):

print("the count is: ",num) num += 1 # num = num + 1

output

the count is: 1 the count is: 2 the count is: 3 the count is: 4 the count is: 5

Note: In while loop always increament|decreament statements must be required for condition becoming false. Otherwise while loop condition not going to be false. So loop not teriminated. It prints infinite times.

Example num=1

while(num <= 5):

print("the count is: ",num)

---->>> Here condition always True.so infinite times executed condition. Q ) WAP to display 0 to n number Square numbers ?

num = int(input('enter any number :')) i = 0

while ( i < num+1 ):

print('Square of '+str(i) +' is '+str(i \*2)) i += 1

**Output:**

enter any number :10 Square of 0 is 0

Square of 1 is 1

Square of 2 is 4

Square of 3 is 9

Square of 4 is 16

Square of 5 is 25

Square of 6 is 36

Square of 7 is 49

Square of 8 is 64

Square of 9 is 81

Square of 10 is 100

Q. WAP to perform sum of digits of a given number ? (123---> 1+2+3 -->> 6) n = int(input('enter any number :'))

sum = 0 while(n!=0):

r = n%10 n = n//10

sum = sum + r

print('The sum of given digit is :',sum)

output

enter any number :123 The sum of given digit is :6

Nested loops:

A loop inside of another loop is called as nested-loop. Example:

for i in range(5):

for j in range(i+1): print(j,end=' ')

print()

output 0

0 1

0 1 2

0 1 2 3

0 1 2 3 4

Transfer Statements

Transfer statements are used to transfer the program control from one location to another location.

We have different types of transfer statements,

1. break
2. continue
3. pass
4. return
5. assert
6. break

---->> break is a keyword which is used only in looping statements.

---->> when ever break occurs it will stop entire iteration and control goes outside the loop.

1. continue

---->> continue is a keyword which is used only in looping statements.

---->> when ever continue occurs it will stop current iteration and executes from next iteration onwords.

For example:

for i in range(1,10): if(i % 2 == 0):

continue if(i % 5 == 0):

break print(i,end=' ')

output 1 3

1. pass

--->> pass is a keyword which is used in program at the time of partial development of code.

Example:

if 10 > 5:

pass

1. return

---->> return is a keyword which is used in functions concept.

---->> if you want to transfer any value in functions then we have to use return keyword.

---->> in functions we can return single value as well as multiple value. Example:

def sum():

a = 10;

b = 20;

c = a + b print(c) return None

result = sum() print("sum is:", result )

output:

30

sum is: None

1. assert

---->> Assert is a keyword which is used to generate exceptions.

---->> If you want to execute certain statements only when the condition is satisfied otherwise program has to stop with error message.

Example:

n = int(input("Enter any number : ")) assert n > 0, "Invalid number" print("Your number is accepted") print("We will process your request ")

Example:

---->> The following example uses the assert statement in the function.

def square(x):

assert x>=0, 'Only positive numbers are allowed' return x\*x

n = square(2) # returns 4

n = square(-2) # raise an AssertionError

Output

Traceback (most recent call last):

assert x > 0, 'Only positive numbers are allowed' AssertionError: Only positive numbers are allowed

**for-else :**

In for-else, a for loop is executed successfully then only else block statements are executed otherwise else block statements will not executed.

For example:

for i in range(1,5):

if(i%5==0):

break print(i,end=' ')

else:

print("working completed") print('ok')

output:

1 2 3 4 working completed ok

#### String Handling Concept:

* A group/sequence of characters is called String.
* Python supports **str** data type to represent string type data.
* String objects are immutable objects that mean we can’t modify the existing string object.
* Insertion order is preserved in string objects.
* Every character in the string object is represented with unique index.
* Python supports both forward and backward indexes.
* Forward index starts with 0 and negative index starts with -1
* Python string supports both "concatenation" and "multiplication" of string objects.
* Strings can be created by enclosing characters inside a single quote or double quotes. Even triple quotes can be used in Python but generally used to represent multiline strings and docstrings.

Quotations in Python:

* Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals, as long as the same type of quote starts and ends the string.
* Generally triple quotes are used to write the string across multiple lines. For example, all the following are legal.

For example1 :

word = 'word'

sentence = "This is a sentence."

Paragraph = """This is a paragraph. It is

made up of multiple lines and sentences."""

Example :

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 |  | 5 |  | 6 | 7 |
| s = S | r | i | n |  | i |  | v | a | s |
| -8 | -7 | -6 | -5 | -4 |  | -3 |  | -2 | -1 |

* If the given index is not available in the string we will get exception like

IndexError

>>> s[8]

**IndexError**: string index out of range.

* If we try to modify the content of string object by using index we will get the

TypeError

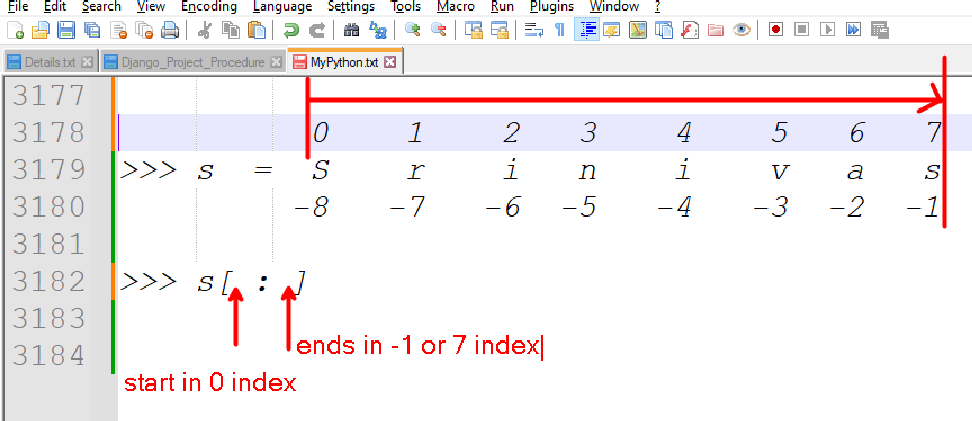
>>> s[2] = 'x'

**TypeError**: 'str' object does not support item assignment

String Slicing :

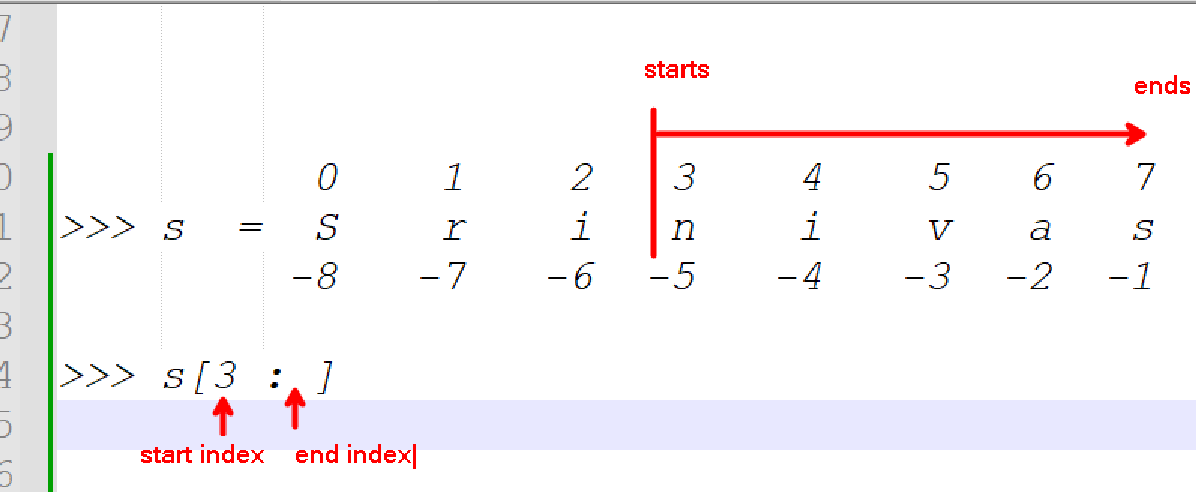
* column(:) is a slice operator , which is used to extract the require content from the given string using given index values.
* [startIndex : endIndex] : Here, start index is 0 position and end index is -1 position.

Example:

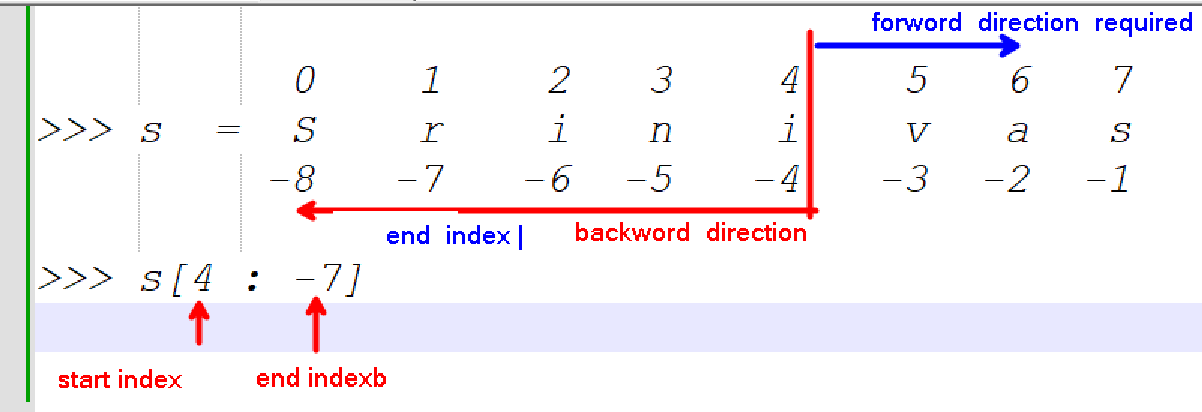
>>> print(x) Srinivas

>>> s[ : ]

Srinivas # Here starts from 0 index and ends with available length

>>> s[ 3 : ] # 'nivas'

* All ways we can slicing the given string as a forward index position only otherwise it returns empty string.

>>> s[ 4 : -7 ] # ' '

String indexing:

To access specific value from a given string by using a given index value is called as indexing.

Syntax: **object[ indexPosition]**

Accessing Values in Strings

>>> s[2] # 'i'

Updating Strings var1 = 'Hello World!'

print("Updated String :- ", var1[:6] + 'Python')

NOTE :

* We can access **individual characters** using "**indexing**" and a **range of characters** using "**slicing**". s[3] , s[2:6]
* Index starts from 0. If we try to access a character out of index range will raise an **IndexError**.

>>> s[15]

**IndexError**: string index out of range

* The index must be an integer. We can't use float or other types, this will result into **TypeError**.

>>> s[1.0]

**TypeError**: string indices must be integers, not 'float'

* The index of -1 refers to the last item, -2 to the second last item and so on.
* We can access a range of items in a string by using the slicing operator (colon).

>>> s [ 2 : 7 ]

'iniva'

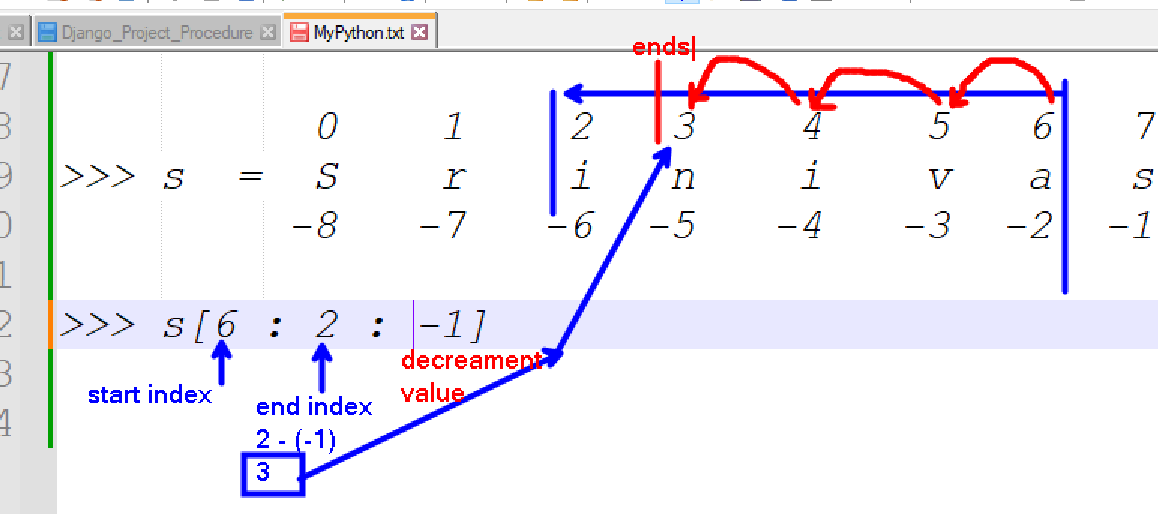
Slicing with step increment:

* + Accessing every second character starting from 0 index to end index

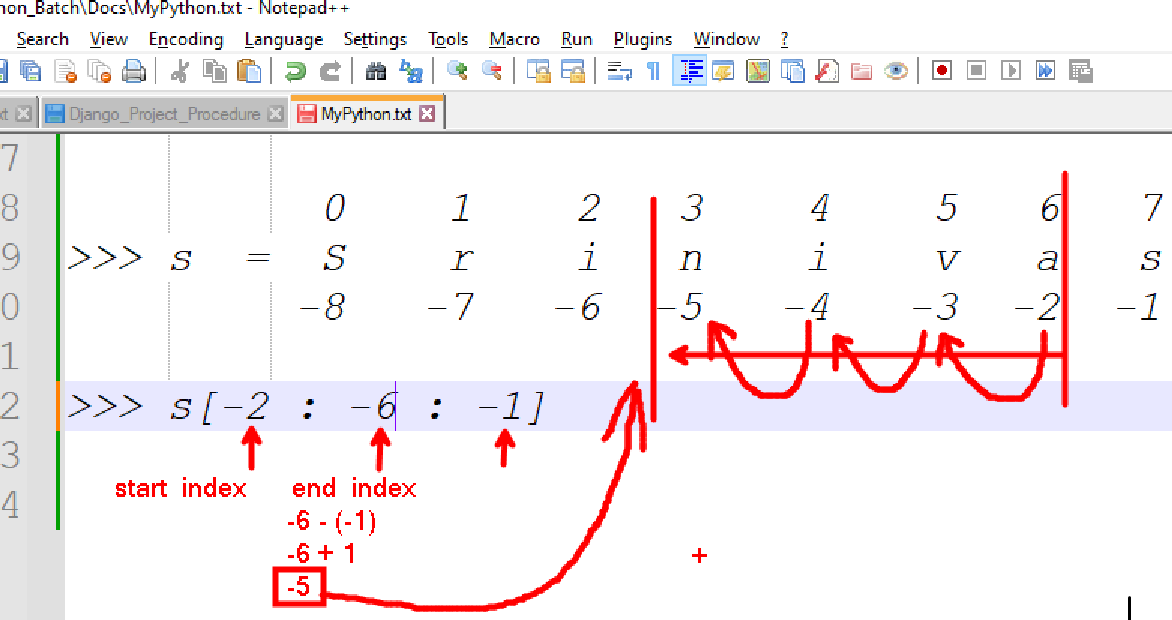
>>> s[ 0 : : 2 ] # 'Siia'

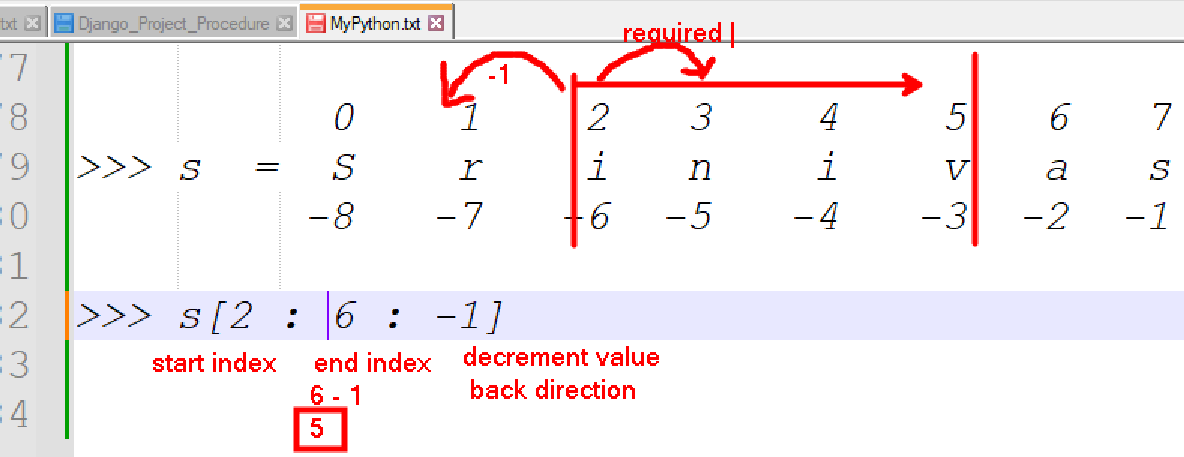
* + Accessing every backword character starting from 4th index to start 0 index

>>> s[4::-1] # 'inirS'

>>> s[6:2:-1] # 'avin'

>>> s[-2:-6:-1] # 'avin'



>>> s[ 2 : 6 : -1 ] # ' '

Concatenation of two or more strings:

* We can concatenate two or more strings into a single one is called concatenation.
* The + operator is used in Python for concatenation. Example:

>>> string1 = 'Python'

>>> string2 = 'Developer'

>>> print( 'String1 + string2 : ', string1 + ' ' + string2 ) Output : String1 + string2 : Python Developer

Multiplication of string:

* Python supports multiplying the given string into n number of times.
* The \* operator can be used to repeat the string for a given number of times.

Example:

>>>string1 = 'Python'

>>>print(string1 \* 3)

Output : PythonPythonPython

String Unpacking

* String unpacking allows extracting string elements automatically.
* String unpacking is the list of variables on the left has the same number of elements as the length of the string.

>>> str1="Python"

>>> print(str1) Python

>>> type(str1) <class 'str'>

>>> id(str1) 23941472

>>> a,b,c,d,e,f = str1 # string unpacking

>>> print(a) P

>>> type(a) <class 'str'>

>>> print(b) y

>>> type(b) <class 'str'>

Membership >>> in

* It Returns True if a given character exists in the given string

>>> s = "Srinivas"

>>> 'r' in s >> True

Membership ---->>> not in

* Returns true if a character does not exist in the given string

>>> ‘S’ in s # True

**Python Operators Interview Questions:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1. | | **What are the main types of operators in Python?** | |  | | | | | | | | | |
|  |  | | | | | | | | | | | | |
| 2. | | **Explain the difference between the == and is operators in Python.** | | | | | |  | | | | | |
|  |  | | | | | | | | | | | | |
| 3. | | **What is the difference between the / and // operators in Python?** | | | | |  | | | | | | |
|  | | | | | | | | | | | | | |
| 4. | | **How are the logical operators and , or , and not used in Python? Provide examples.** | | | | | | | | | | |  |
|  |  | | | | | | | | | | | | |
| 5. | | **How do you overload operators in Python?** |  | | | | | | | | | | |
|  |  | | | | | | | | | | | | |
| 6. | | **Explain the purpose of the in and not in operators in Python.** | | | |  | | | | | | | |
|  |  | | | | | | | | | | | | |
| 7. | | **What is the purpose of the identity operators is and is not?** | | |  | | | | | | | | |
|  |  | | | | | | | | | | | | |
| 8. | | **What are assignment expressions (walrus operator :=)? When and how are they used?** | | | | | | | | | |  | |
|  |  | | | | | | | | | | | | |
| 9. | | **How can you use the \* and \*\* operators for packing and unpacking in Python?** | | | | | | | | |  | | |
|  |  | | | | | | | | | | | | |
| 10. | | **What is operator precedence in Python? How is it determined?** | | |  | | | | | | | | |
|  |  | | | | | | | | | | | | |
| 11. | | **How can you compare two lists in Python?** |  | | | | | | | | | | |
|  |  | | | | | | | | | | | | |
| 12. | | **What are the membership operators in Python? Provide examples of their usage.** | | | | | | | |  | | | |
|  |  | | | | | | | | | | | | |
| 13. | | **What are the identity operators in Python? Provide examples of their usage.** | | | | | | |  | | | | |



**Python Control Statements Interview Questions:**

|  |  |  |
| --- | --- | --- |
| 1. **What are the different types of control statements in Python?** 2. **Explain the usage of if...else statements in Python.** 3. **What is the purpose of the elif statement in Python?** 4. **How does Python handle nested control statements?** 5. **Write a Python program to check if a given number is positive, negative, or zero.** 6. **Write a Python program to find the largest of given Three numbers.** 7. **Write a Python program to check if a year is a leap year or not.** 8. **Write a Python program to check if a given number is even or odd.** 9. **Write a Python program to determine the grade of a student based on their percentage.** | | |
| **10.** | **Write a Python program to check whether a given number is prime or not.** |  |



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Python Looping or Iterative Statements Interview Questions:** | | |  |
|  | **1. Explain the difference between the 'while' and 'for' loop in Python.** | | | |
|  | | |  | |
|  | **2. How can you terminate a loop prematurely in Python?** | | | |
|  | | |  | |
|  | **3. What is the 'break' statement used for in Python?** | | | |
|  | |  | | |
| **4. Describe the 'continue' statement and its usage.** | | | | |

**5. How can you iterate over a sequence while keeping track of the index?**

**6. What is the purpose of the 'pass' statement in Python?**

**7. What is the purpose of the 'return' statement in Python?**

**8. What is the purpose of the 'assert' statement in Python?**

**9. What is the purpose of the 'for - else' statement in Python?**

|  |  |  |
| --- | --- | --- |
| **10.** | **Write a Python program to print all the numbers from 1 to 10.** |  |

|  |  |  |
| --- | --- | --- |
| **11.** | **Write a Python program to find the sum of all numbers from 1 to 100.** |  |

|  |  |  |
| --- | --- | --- |
| **12.** | **Write a Python program to find the factorial of a number using a while loop.** |  |

|  |  |  |
| --- | --- | --- |
| **13.** | **Write a Python program to generate the Fibonacci sequence up to n terms.** |  |

|  |  |  |
| --- | --- | --- |
| **14.** | **Write a Python program to reverse a given integer.** |  |

|  |  |  |
| --- | --- | --- |
| **15.** | **Write a Python program to print the multiplication table of a given number.** |  |
|  | | |

### String Functions or Methods:

Python supports so many functions or methods in String concept to handling the String data. They are ,

1. capitalize():
   * This function converts first letter of first word in the given string into upper case format.

>>> str1 = 'python developer'

>>> str1.capitalize() # 'Python developer'

1. title():
   * This function converts first character of each word in the given string into upper case format.

>>> str1 = 'python developer'

>>> str1.title() # 'Python Developer'

1. islower():
   * This function checks whether the given string contains all lower case letters or not. If all are lower case then it will return True else False.

>>> str1='python developer'

>>> str1.islower() # True

>>> str2="Python"

>>> str2.islower() # False

1. isupper():
   * This function checks whether the given string contains all upper case letters or not. If all are upper case then it will return True else False.

>>> str1='python developer'

>>> str1.isupper() # False

>>> str3='PYTHON'

>>> str3.isupper() # True

1. lower():
   * This function converts all letters of given string into lower case.

>>> str3='PYTHON'

>>> str3.lower() 'python'

1. upper():
   * This function converts all letters of given string into upper case.

>>> str1='python developer'

>>> str1.upper() 'PYTHON DEVELOPER'

1. len():
   * This function counts the number of characters in the given string and returns count value.

>>> str1='python developer'

>>> len(str1) 16

1. count(element, start\_index\_position, end\_index\_position):
   * This function counts number of occurrences of a specific character in the given string and returns count value.
   * here start\_index\_position value is starts with 0 by default and end\_index\_position ends with -1 by default.

>>> str1='python developer'

>>> str1.count('o') 2

>>> str1.count('o',5) 1

1. find(element, start\_index\_position, end\_index\_position):

Syntax : str.find(element , index\_position)

* + This function finds the index position of specific character in the given string.
  + By default index position value is taking zero (0), means element searching from 0 index place onwords.
  + If we specifiying any index value then searching starts from that index value only.
  + If searching element not available in given string then it returns -1 value.

>>> str1='python developer'

>>> str1.find('o') 4 >> for first occurrence of ‘o’

>>> str1.find('o', 5 ) 12 >> for second occurrence of ‘o’

1. index(element, start\_index\_position, end\_index\_position):
   * The index() method finds the first occurrence of the specified value.
   * The index() method raises an exception if the value is not found.
   * The index() method is almost the same as the find() method, the only difference is that the find() method returns -1 if the value is not found.

Syntax : string.index(element\_value, start\_index\_value, end\_index\_value)

Parameter Description

value Required. The value to search for

start Optional. Where to start the search. Default is 0

end Optional. Where to end the search. Default is to the end of the string

1. split():
   * This function splits the given string into multiple strings and returns in the form of list of strings.
   * By default space is taken as splitting parameter.

>>> str1='python developer'

>>> str1.split() ['python', 'developer']

* + If you want to splitting the string by using any special parameter insted of space then we use that special parameter using split().

>>> str1='python developer'

>>> str1.split('o') ['pyth', 'n devel', 'per']

1. splitlines()
   * By using this **splitlines()** method we can find total number of rows from a given string and it returns in the form of list and every single line as a one strings .
   * It is used \n as a separator.

>>> s1 = """Python is esay. Python is more simple.

Python is a language """

>>> print(s1.splitlines())

['Python is esay.', 'Python is more simple.', 'Python is a language']

>>> print(len(s1.splitlines())) 3

1. swapcase():
   * This function swaps all lower case letters into upper case and swaps all upper case letters into lower case letters.

>>> str1='PyThOn'

>>> str1.swapcase() 'pYtHoN'

1. reversed():
   * This function reverses the given string and returns reversed object only but not values directly.
   * If you want to get the reversed string for given string then we use join() method.
   * reversed() method is not done any changes in the given main string and result will store into a separate variable.

>>> str='Python'

>>> print(str) Python

>>> str1=''.join(reversed(str))

>>> print(str1) nohtyP

>>> s

'Srinivas'

>>> reversed(s)

<reversed object at 0x0042D5D0>

# it prints reversed object only. It creates new object. use join() to get reverse string.

>>> ''.join(reversed(s)) 'savinirS'

1. sorted(obj) :
   * It will sorting the characters in assending order by default.

>>> st = 'Python'

>>> sorted(st)

['P', 'h', 'n', 'o', 't', 'y']

* + To get as a string format then use join() like below,

>>> ''.join(sorted(st)) 'Phnoty'

Q ) How to display the given string in descending order?

>>> st="python"

>>> ''.join(reversed(sorted(st))) 'ytponh'

or

>>> ''.join(sorted(st,reverse=True)) 'ytonhP'

Q ) How to display the given string with two dots between each character?

>>> s1='python'

>>> s10='..'.join(s1)

>>> s10

'p..y..t..h..o..n'

1. replace():
   * This function replaces an existing character(s) with new character(s). syntax : str.replace(old\_sub\_string , new\_sub\_string)

>>> str1='python learner'

>>> print(str1) python learner

>>> str2=str1.replace('learner','developer')

>>> print(str2) python developer

>>> print(str1) python learner

Note: we can remove any character(s) with non-empty space.

>>> str1='Python'

>>> print(str1) Python

>>> type(str1) <class 'str'>

>>> str2 = str1.replace('thon','')

>>> print(str2) Py

>>> type(str2) <class 'str'>

##### del :

* + We can also remove string object by using "del" command, but we can not delete string object elements(charecters) because string is immutable object.

Example:

>>> str1="Python Srinivas"

>>> print(str1) Python Srinivas

>>> type(str1) <class 'str'>

>>> id(str1) 63806464

>>> del str1 #deleting str1

>>> print(str1) #after deleting **NameError**: name 'str1' is not defined **Note:**

>>> del str1[0]

**TypeError**: 'str' object doesn't support item deletion

1. format() :
   * The format() method formats the specified value(s) and insert them inside the string's placeholder.
   * The placeholder is defined using curly brackets: {}.
   * The placeholders can be identified using named indexes {price}, numbered indexes {0}, or even empty placeholders {}.
   * The format() method returns the formatted string.

Syntax : string.format(value1, value2...) Example:

>>> txt1 = "My name is {fname}, My age is {age}".format(fname = "Srinivas", age = 30)

>>> txt1

"My name is Srinivas, My age is 30"

>>> a = "Employee name is {0} and age is {1}".format(20,'Kiran')

>>> a

'Employee name is 20 and age is Kiran'

>>> a = "Employee name is {1} and age is {0}".format(20,'Kiran')

>>> a

'Employee name is Kiran and age is 20'

>>> name = 'Ramu'

>>> age = 30

>>> a = "Employee name is {} and age is {}".format('Ravi',40)

>>> a

'Employee name is Ravi and age is 40'

When Errors Coming:

a = "Employee name is {name} and age is {age}".format(age,name)

**Outpput : KeyError**: 'name'

a = "Employee name is {0} and age is {1}".format(age=20,name='Kiran')

**Output** : **IndexError**: tuple index out of range

1. endswith():
   * The endswith() method returns True if the string ends with the specified value, otherwise False.

**syntax** : string.endswith(value, start\_index\_value, end\_index\_value)

f’string concept’

=============

>>> ename="Virat"

>>> age =30

>>> f'Employee name is {ename} and age is {age}' 'Employee name is Virat and age is 30'

Q. Check if the string ends with a punctuation sign (.):

>>> txt = "Hello, welcome to my world."

>>> x = txt.endswith(".")

>>> x

True

1. startswith() :
   * Returns true if the string starts with the specified value

>>> txt = "Hello, welcome to my world."

>>> x = txt.startswith("wel", 7, 20)

>>> x

True

# use a dictionary with ascii codes to replace 83 (S) with 80 (P):

>>> mydict = {83: 80};

>>> txt = "Hello Sam!";

>>> print(txt.translate(mydict));

1. strip():
   * The strip() method removes any leading (spaces at the beginning) and trailing (spaces at the end) characters (space is the default leading character to remove)

>>> txt = " banana "

>>> x = txt.strip()

>>> print("of all fruits", x, "is my favorite") of all fruits banana is my favorite

1. **rstrip() :** Returns a right trim version of the string
2. **lstrip() :** Returns a left trim version of the string

**Questions:**

### Python String Concept

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **1.** | **What are strings in Python? How are they represented?** | |  | |
| 1. **How do you create a Multiline string in Python?** 2. **How do you create a Single line string in Python?** | | | | |
| **4. How do you access characters in a string in Python?** | |  | | |
|  | | | | |
| **5. How do you check the length of a string in Python?** | |  | | |
| 1. **What is string slicing in Python?** 2. **How do you concatenate strings in Python?** 3. **How do you repeated strings in Python?** 4. **What are some common string methods in Python?** 5. **What is string formatting in Python?** 6. **How do you reverse a string in Python?** | | | | |
| **12. How do you check if a string is a palindrome in Python?** | | | |  |
| **13. What is string interpolation in Python? (i.e., str.format(), f” ” )** | | | | |

**TUPLE- DATA Type :**

* + Tuple is used to represent a set of homogeneous or heterogeneous elements into a single entity.
  + Tuple objects are immutable that means once if we create a tuple object later we cannot modify those tuple elements.
  + All elements are separated by commas (,) and enclosed by parentheses. Parentheses are optional. ()
  + Tuple allows duplicate elements.
  + Every element in the tuple has its own index number
  + Tuple supports both forward indexing and also backward indexing, forward indexing starts from 0 and backward indexing starts from -1.
  + If we take **only one element** in the tuple then we should use **comma (,)** after that single element.

**t = (10,)** --->> tuple type t1 = (10) --->> int type

* + Tuples can be used as keys to the dictionary.
  + We can create a tuple in different ways, like with tuple(), with () or without () also.
  + The main difference between lists and tuples is- Lists are enclosed in square brackets like [] and their elements and size can be changed, while tuples are enclosed in parentheses like () and their elements and size cannot be updated.

Creating a tuple with tuple() :

>>> tup = tuple([10,20,30,True,'Python'])

>>> print(tup) (10, 20, 30, True, 'Python')

>>> type(tup) <class 'tuple'>

>>> id(tup) 52059760

Creating an empty tuple:

Example:

>>> tup = () #creating empty tuple

>>> print(tup) ()

>>> type(tup) <class 'tuple'>

>>> id(tup) 23134256

Creating a tuple with ()

Example:

>>> tup2 = (10,20,30,40,50) #creating homogeneous tuple

>>> print(tup2) (10, 20, 30, 40, 50)

>>> type(tup2) <class 'tuple'>

>>> id(tup2) 63484864

Creating a tuple without ()

Example:

>>> tup = 10,20,True,'Py' #creating tuple without parenthesis

>>> print(tup) (10, 20, True, 'Py')

>>> type(tup) <class 'tuple'>

>>> id(tup) 67086688

Creating a tuple with heterogeneous elements

Example:

>>> tup1 = (10,20,30,True,"Python",10.5,3+5j) #creating heterogeneous tuple

>>> print(tup1)

(10, 20, 30, True, 'Python', 10.5, (3+5j))

Creating a tuple with homogeneous elements

Example:

>>> t = (10,20,30,40) # creating homogeneous tuple

>>> print(t) # (10,20,30,40)

NOTE : **tuple with Single value**

* + Creating a tuple with a single element is tricky, if we take only one element then the type of that tuple will be based on specified element type.

>>> t2 = (1)

>>> t2 1

>>> type(t2) <type 'int'>

>>> t2 = (True)

>>> print(t2) True

>>> type(t2) <type 'bool'>

>>> t2 = ('a')

>>> print(t2) 'a'

>>> type(t2) <type 'str'>

Solution :

So to solve the above problem we should use comma (,) after the element in the tuple if tuple contains single element.

For example:

>>> t2=(1,)

>>> print(t2) (1,)

>>> type(t2) <type 'tuple'>

>>> t2=(False,)

>>> print(t2) (False,)

>>> type(t2) <type 'tuple'>

>>> t2 = ('a')

>>> print(t2) ('a',)

>>> type(t2) <type 'tuple'>

Tuple Indexing:

Tuple indexing is nothing but fetching a specific element from the existing tuple by using its index value.

Tuple Slicing:

Tuple slicing is nothing but fetching a sequence of elements from the existing tuple by using their index values.

Example:

>>>tup = (10,20,30,True,"Python",10.5,3+5j,10)

>>> print(tup) # (10, 20, 30, True, 'Python', 10.5, (3+5j), 10)

>>> type(tup) # <class 'tuple'>

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| tup = ( 10, | 20, | 30, | True, | "Python", | 10.5, | 3+5j, | 10 ) |
| -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1 |

>>> tup[0:4] (10, 20, 30, True)

>>> tup[0:0] ()

>>> tup[0:1] (10,)

>>> tup[0:5] (10, 20, 30, True, 'Python')

>>> tup[3:5] (True, 'Python')

>>> tup[2:-2] (30, True, 'Python', 10.5)

>>> tup[-5:-2] (True, 'Python', 10.5)

>>> tup[-5:] (True, 'Python', 10.5, (3+5j), 10)

>>> tup[6:] ((3+5j), 10)

Tuple concatenation :

* + We can concatenate two or more tuples in python. Example:

>>> tup1=(1,2,3,'a',True) #creating first tuple tup1

>>> print(tup1) (1, 2, 3, 'a', True)

>>> type(tup) <class 'tuple'>

>>> tup2=(10,20,False,'b') #creating second tuple tup2

>> print(tup2) (10, 20, False, 'b')

>>> type(tup2) <class 'tuple'>

>>> tup3 = tup1+tup2 #concatenating tup1 and tup2 as tup3

>>> print(tup3) (1, 2, 3, 'a', True, 10, 20, False, 'b')

>>> type(tup3) <class 'tuple'>

Tuple multiplication or repetition :

* + We can multiply or repeat a tuple n number of times.

>>> tup1=(1,2,3,'a',True)

>>> print(tup1) (1, 2, 3, 'a', True)

>>> type(tup1) <class 'tuple'>

>>> tup1\*3

(1, 2, 3, 'a', True, 1, 2, 3, 'a', True, 1, 2, 3, 'a', True)

Tuple Data type Methods :

1. len():
   * This function returns no.of elements in the tuple.

>>> tup = (1,2,3,4,'a',5.5)

>>> len(tup) 6

1. count():
   * This function counts the number of occurences of a specific elements. This function takes exactly one argument like element.

Example:

>>> tup = (1,10,20,True,0)

>>> tup.count(1) 2

>>> tup.count(0) 1

1. index(object, index\_value,end\_index):
   * This function is used to find the index value of specific|given element.
   * This function returns by default first occurence of given element index\_value.
   * It is also accepts the second parameter as index value, it is used for from where you want search the given index. By default index\_value starts from zero.

Example:

>>> tup=(1,10,20,True,0)

>>> tup.index(0) 4

>>> tup.index(10) 1

>>> tup.index(20) 2

1. max():
   * This function returns maximum value from the tuple elements. Example:

>>> tup=(1,3,2,55,3,5,23)

>>> max(tup) 55

1. min():
   * This function returns minimum value from the tuple elements. Example:

>>> tup=(1,3,2,55,3,5,23)

>>> min(tup) 1

1. sum():
   * this function returns sum of all the elements. Example:

>>> tup=[1,9,5,11,2]

>>> sum(tup) 28

1. sorted(object):
   * sorted() is going to take the elements from given object and arranging all the elements by default in a assending order.
   * after arranging all the elements in assending order then resoult store in a new variable.
   * sorted() method is not doing any changes in a original object and the result store in a new object.
   * sorted() method returns result in a list format by defalt.
   * if you want to get in tuple format then use tuple() method Example:

>>> tup = (1,3,2,55,3,5,23)

>>> sorted(tup) [1, 2, 3, 3, 5, 23, 55]

Note: by default this function sorts the data in ascending order. We can also get in descending order by setting True for reverse.

Example:

>>> tup=(1,3,2,55,3,5,23)

>>> sorted(tup,reverse=True) [55, 23, 5, 3, 3, 2, 1] Or

>>> t1 = tuple([1,2,3,7,4])

>>> t1 (1, 2, 3, 7, 4)

>>> t2 = reversed(t1)

>>> tuple(t2) (4, 7, 3, 2, 1)

1. reversed():
   * reversed() is going to take the elements from given object and arranging all the elements by default in a reversing order.
   * after arranging all the elements in reversing order then resoult store in a new variable.
   * reversed() method is not doing any changes in a original object and the result store in a new object.
   * reversed() method returns result in a <reversed object at 0x03EFFC30> format by defalt. Internally elements are reversed.
   * if you want to get in tuple format then use tuple() method

>>> t = (10, 40, 60, 20)

>>> t2 = reversed(t)

>>> t2

<reversed object at 0x03EFFC30>

>>> tuple(t2) # (20, 60, 40, 10)

Note:

* + tuple object is not supporting both sort() and reverse() and copy() and clear() also.

>>> t = (10, 40, 60, 20)

>>> t2 = sort(t)

**NameError:** name 'sort' is not defined

>>> t3 = reverse(t)

NameError: name 'reverse' is not defined

>>> t = (10, 40, 60, 20)

>>> id(t) # 65890704

>>> t2 = t

>>> print(t2) # (10, 40, 60, 20)

>>> id(t8) # 65890704

>>> t2 = t.copy()

AttributeError: 'tuple' object has no attribute 'copy'

DEL Command :

We cannot delete the elements of existing tuple but we can delete the entire tuple object by using del command.

Example:

>>> tup = (10,20,"Python",1.3)

>>> print(tup) # (10, 20, 'Python', 1.3)

>>> type(tup) # <class 'tuple'>

>>> del tup # deleting tuple by using del command.

>>> print(tup) # after deleting

**NameError**: name 'tup' is not defined

We can replace the elements of list but not tuple, like

>>> lst=[10,20,30,'Py',True]

>>> lst[4]=False # it is possible in list

>>> print(lst) [10, 20, 30, 'Py', False]

>>> tup = (10,20,30,'Py',True)

>>> tup[4]=False # it is not possible in tuple **TypeError**: 'tuple' object does not support item assignment Tuple packing:

* + We can create a tuple by using existing variables, so its called tuple packing.

>>> a=10

>>> b=20

>>> c='Python'

>>> d=2+5j

>>> tup=(a,b,c,d)

>>> print(tup) (10, 20, 'Python', (2+5j))

>>> type(tup) <class 'tuple'>

>>> id(tup) 62673808

Tuple Unpacking

* + Tuple unpacking allows to extract tuple elements automatically.
  + Tuple unpacking is the list of variables on the left has the same number of elements as the length of the tuple

>>> tup=(1,2,3,4)

>>> a,b,c,d=tup # tuple unpacking

>>> print(a) 1

>>> print(b) 2

>>> print(c) 3

>>> print(d) 4

Nested tuple:

* + Python supports nested tuple, means a tuple in another tuple.
  + Tuple allows list as its element. Example:

>>> t1=(1,'a',True)

>>> print(t1) (1, 'a', True)

>>> type(t1) <class 'tuple'>

>>> t2=(10,'b',False)

>>> print(t2) (10, 'b', False)

>>> type(t2) <class 'tuple'>

>>> t3=(t1,100,'Python',t2) # creating a tuple with existing tuples t1 and t2

>>> print(t3) ((1, 'a', True), 100, 'Python', (10, 'b', False))

>>> type(t3) <class 'tuple'>

>>> print(t3[0]) (1, 'a', True)

>>> print(t3[1]) 100

>>> print(t3[2]) Python

>>> print(t3[3]) (10, 'b', False)

>>> print(t3[3][0]) 10

>>> print(t3[3][1]) b

>>> print(t3[3][2]) False

>>> print(t3[0][0]) 1

>>> print(t3[0][1]) a

>>> print(t3[0][2]) True

>>> t3[0:2] ((1, 'a', True), 100)

>>> t3[2:4] ('Python', (10, 'b', False))

>>> t3[-2:4] ('Python', (10, 'b', False)) Note:

* We can’t modify any element of the above tuples because tuples are immutable.
* If the tuple contains a list as a element then we can modify the elements of the list as it a mutable object.

Example:

>>> tup = (1,2,[10,12,'a'],(100,200,300),3,'Srinivas')

>>> print(tup) # (1, 2, [10, 12, 'a'], (100, 200, 300), 3, 'Srinivas')

>>> tup[0] 1

>>> tup[1] 2

>>> tup[2] [10, 12, 'a']

>>> tup[3] (100, 200, 300)

>>> tup[4] 3

>>> tup[5] 'Srinivas'

>>> tup[0]=50

# trying to replace element 1 with 50, interpreter throws error. TypeError: 'tuple' object does not support item assignment

>>> tup[2][0]=50

# trying to replace element of list 10 with 50, interpreter accepts.

>>> print(tup) (1, 2, [50, 12, 'a'], (100, 200, 300), 3, 'Srinivas')

Converting tuple to list :

>>> tup=(1,2,4,9,8) #creating a tuple

>>> print(tup) (1, 2, 4, 9, 8)

>>> type(tup) <class 'tuple'>

>>> lst = list(tup) # converting tuple to list by using list()

>>> print(lst) [1, 2, 4, 9, 8]

>>> type(lst) <class 'list'>

Converting list to tuple:

>>> lst=[10,20,30,40,'a'] #creating a list

>>> print(lst) [10, 20, 30, 40, 'a']

>>> type(lst) <class 'list'>

>>> tup=tuple(lst) #converting list to tuple by using tuple()

>>> print(tup) (10, 20, 30, 40, 'a')

>>> type(tup) <class 'tuple'>

Converting tuple to string:

>>> tup=('a','b','c') #creating tuple

>>> print(tup) ('a', 'b', 'c')

>>> type(tup) <class 'tuple'>

>>> str1=''.join(tup) #converting tuple to string by using join method

>>> print(str1) abc

>>> type(str1) <class 'str'>

Converting string to tuple:

>>> str1="Python Srinivas" #creating a string

>>> print(str1) Python Srinivas

>>> type(str1) <class 'str'>

>>> tup=tuple(str1) #converting a string by using tuple function.

>>> print(tup)

('P', 'y', 't', 'h', 'o', 'n', ' ', 'S', 'r', 'i', 'n', 'i', 'v', 'a', 's')

>>> type(tup) <class 'tuple'> Note:

>>> t = ("a","b","c",10)

>>> ''.join(t)

**TypeError**: sequence item 3: expected str instance, int found

Advantages of Tuple over List:

* Generally we use tuple for heterogeneous elements and list for homogeneous elements.
* Iterating through tuple is faster than list because tuples are immutable, So there might be a slight performance boost.
* Tuples can be used as key for a dictionary. With list, this is not possible because list is a mutable object.
* If you have data that doesn't change, implementing it as tuple will guarantee that it remains write-protected.

#### LIST - DATA STRUCTURE:

* + A list is a collection of elements. These elements may be homogeneous(same types) or heterogeneous (diffrent types).
  + A list also allows duplicate elements.
  + Insertion order is preserved in list.
  + List elements are separated by commas and enclosed within square brackets [10,20,'abc'].
  + Every element in the list has its own unique index number.
  + List supports both forward indexing and backward indexing, forward index starts from 0 and backward index starts from -1.
  + We access either specific element by using "indexing" or set of elements by using "slicing" from the List.

For example, l[0] --->> indexing l[0:3] >> slicing

* + We can create list in different ways.
    1. By using "list()" function,
    2. By using square brackets []
    3. By using range() function.
  + List objects are mutable means we can change the list elements.

Creating List by using list():

1. This list() allows only one string value with set of characters.
2. If we give int type data in the list() function then interpreter will throw ‘**TypeError’** error.

Example:

>>> List1 = list() #creating empty list

>>> print(List1) []

>>> type(List1) <class 'list'>

>>> List1 = list('python') #creating list with set of characters

>>> print(List1) ['p', 'y', 't', 'h', 'o', 'n']

>>> type(List1) <class 'list'>

Creating list by using square brackets []

>>> List1 = [] #creating empty list

>>> print(List1) []

>>> type(List1) <class 'list'>

>>> List1 = [1,2,3,4,5] #creating list with homogeneous elements

>>> print(List1) [1, 2, 3, 4, 5]

>>> type(List1) <class 'list'>

>>> List1 = [10,11,'Python',5.5,True,2+3j] #creating list with heterogeneous elements

>>> print(List1) [10, 11, 'Python', 5.5, True, (2+3j)]

>>> type(List1) <class 'list'>

Creating list by using range() function:

We can also use range() function to create list.

Syntax : range( StartingIndexValue, LastValue-1, RangeValue ) For example: range( 10)

* 1. Here, both StartingIndexValue and RangeValue are optional.
  2. The default StartingIndexValue is 0.
  3. The default RangeValue is 1.

Example:

>>> List1 = list(range(10))

>>> print(List1) [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> type(List1) <class 'list'>

>>> List1 = list(range(0,10))

>>> print(List1) [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> type(List1) <class 'list'>

>>> List1 = list(range(2,8))

>>> print(List1) [2, 3, 4, 5, 6, 7]

>>> type(List1) <class 'list'>

List Indexing

1. By using list indexing we can fetch specific element from the list.
2. It supports both forward and backward indexing

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |  | |
| List1 =[ | 10, | 20, | 30, | 'Python', | True, |  | 1.5, | 2+3j] |

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

Example:

>>> List1 = [10,20,30,'Python',True,1.5,2+3j]

>>> print(List1[0]) 10

>>> print(List1[1]) 20

List Slicing

1. By using slicing we can fetch set of characters from list.
2. It also supports both forward and backward indexing
3. Colon (:) is the slicing operator. Example:

>>> List1 = [10,20,30,'Python',True,1.5,2+3j]

>>> print(List1[0:2]) [10, 20]

>>> print(List1[2:5]) [30, 'Python', True]

NOTE : List is a "mutable" object that means we can update or replace the existing list with new elements. But id reference value is not changed.

List Concatenation:

Python supports concatenating two or more lists into single list. Example:

>>> list1 = [10,20,50]

>>> list2 = [70,10,20,50]

>>> list1 + list2 # [10, 20, 50, 70, 10, 20, 60]

List Multiplication or List Repetition

Python supports multiplying the given list into N number of times. Example:

>>> List1 = [10,'Python',5.5]

>>> List2 = List1\*3

>>> print(List2) # [10, 'Python', 5.5, 10, 'Python', 5.5, 10, 'Python', 5.5]

### List functions or methods:

1. len():
   * This function counts number of elements in the list. Example:

>>> List1 = [10,20,'Python',30,True,'Narayana',10,3+4j]

>>> len(List1) # 8

1. count(element):
   * This function counts the no.of occurrences of specific element in the list. Example:

>>> List1 = [10,20,'Python',30,True,'Narayana',10,3+4j]

>>> List1.count(10) 2

[Example:](#_TOC_250002)

>>> lst = [1,2,3,True,False,1,0,False]

>>> lst.count(1) 3

[Example:](#_TOC_250001)

>>> lst = [1,2,3,True,False,1,0,False,1+9j,1.1]

>>> lst.count(0) 3

>>> lst.count(False) 3

1. index(object, index\_value):
   * This function finds the index value for specific element. Example:

>>> List1 = [10,20,'Python',30,True,'Narayana',10,3+4j]

>>> List1.index(10) 0

[Example:](#_TOC_250000)

>>> lst = [1,2,3,True,False,1,0,False,1+9j,1.1]

>>> lst.index(1,1) 3

Example:

>>> List1 = [10,20,'Python',30,True,'Narayana',10,3+4j]

>>> List1.index(10,1) # 6

Index on nested list:

>>> lst = [100, True, 1, 2, 3, 4, 5, 0, [8, 9], 10, 20, 4]

>>> lst[8].index(9) # 1

>>> lst[8].index(8) # 0

0 1 2 3 4 5 6 7 8 9 10 11

lst = [ 100, True, 1, 2, 3, 4, 5, 0, [8, 9], 10, 20, 4 ]

-12 -11 -10 -9 -8 -7 -6 -5 -4 -3 -2 -1

1. append():
   * This function adds new element at the end of the existing list. Example:

>>>List1 = [10,20,'Python',30,True,'Srinivas',10,3+4j]

>>> List1.append(50)

>>> print(List1) # [10, 20, 'Python', 30, True, 'Srinivas', 10, (3+4j), 50] Example:

>>> List1 = [10,20,'Python',30,True,'Srinivas',10,3+4j]

>>> List1.append(60)

>>> print(List1) # [10, 20, 'Python', 30, True, 'Srinivas', 10, (3+4j), 60]

NOTE:

* + We can also add multiple elements in the list by using append method but those multiple elements work like nested list or sub list in the existing list and also getting single index value for this sub list

>>> List1 = [10,20,'Python',30,True,'Srinivas',10,3+4j]

>>> List1.append([0,1,2])

>>> print(List1)

[10, 20, 'Python', 30, True, 'Srinivas', 10, (3+4j), [0, 1, 2]]

1. extend():
   * This function adds multiple elements at the end of the existing list as multiple elements

Example:

>>>List1=[10,20,'Python',30,True,'Srinivas',10,3+4j]

>>> List1.extend([70,80,90])

>>> print(List1)

[10, 20, 'Python', 30, True, 'Srinivas', 10, (3+4j), 70, 80,90]

1. insert(index\_position, object):
   * This function adds an element at any required index place in the existing list. Example:

>>>List1=[10,20,'Python',30,True,'Srinivas',10,3+4j]

>>> List1.insert(2,100)

>>> print(List1)

[10, 20, 100, 'Python', 30, True, 'Srinivas', 10, (3+4j)] Example:

>>> List1=[10,20,'Python',30,True,'Srinivas',10,3+4j]

>>> List1.insert(3,'Durga')

>>> print(List1)

[10, 20, 'Python', 'Durga', 30, True, 'Srinivas', 10, (3+4j)]

* + We can also add multiple elements in the list at required place by using insert method but those multiple elements work like nested list or sub list in the existing list and it contains single index value.

>>> List1=[10,20,'Python',30,True,'Narayana',10,3+4j]

>>> List1.insert(0,[0,1,2,3])

>>> print(List1)

[[0, 1, 2, 3], 10, 20, 'Python', 30, True, 'Srinivas', 10, (3+4j)]

1. remove(object):
   * This function removes specific element in the existing list.
   * This function allows one argument and that should be element name.
   * search for removing element from 0 index onwords and removing first occurence.

Example:

>>> List1 = [10,20,'Python',30,True,'Srinivas',10,3+4j]

>>>List1.remove(10)

>>> print(List1)

[20, 'Python', 30, True, 'Srinivas', 10, (3+4j)]

Example:

>>> List1 = [10,20,'Python',30,True,'Srinivas',10,3+4j]

>>> List1.remove(True)

>>> print(List1)

[10, 20, 'Python', 30, 'Srinivas', 10, (3+4j)]

Example:

>>> l = [1, 2, 3, 1, 3]

>>> l

[1, 2, 3, 1, 3]

>>> l.remove(1) # only first occurence element deleted

>>> l

[2, 3, 1, 3]

>>> l.remove(6)

**ValueError**: list.remove(x): x not in list

Note : if value is not in given list then it returns ValueError

>>> l = [10, 30, 10]

>>> l.remove()

**TypeError**: remove() takes exactly one argument (0 given)

1. pop(index):
   * This function removes specific element from given list.
   * This function also allows only one argument and that should be index value for an element.
   * If index value not given then by default last element removed automatically from given list and also returns this removed value.

Example:

>>> List1 = [10,20,'Python',30,True,'Srinivas',10,3+4j]

>>> List1.pop(1) # 20

>>> print(List1) # [10, 'Python', 30, True,'Srinivas', 10, (3+4j)] Example:

>>> List1 = [10,20,'Python',30,True,'Srinivas',10,3+4j]

>>> List1.pop(2) # 'Python'

>>> print(List1) # [10, 20, 30, True, 'Srinivas', 10, (3+4j)] Example:

>>> l = [10,30,60,10,40]

>>> l.pop() 40

>>> l

[10, 30, 60, 10]

>>> l.pop(2) # index position 2 value removed and returns also. 60

>>> l

[10, 30, 10]

Example:

>>> l = [10, 30, 10]

>>> id(l) 67974328

>>> l2 = l.pop() # removed value store in l2.

>>> l [10, 30]

>>> id(l) 67974328

>>> l2 10

>>> id(l2) 1834313024

1. reverse():
   * This function reverses the existing list elements. Example:

>>> List1 = [10,20,'Python',30,True,'Srinivas',10,3+4j]

>>> List1.reverse()

>>> print(List1) # [(3+4j), 10, 'Srinivas', True, 30, 'Python', 20, 10] Example:

>>> l = [10,40,60,20,30,40]

>>> l2 = l.reverse() # reverse the elements in given list only but not assign to l2

>>> l2

>>> # no output returns l2.

>>> l

[40, 30, 20, 60, 40, 10]

>>>

1. reversed(object) :
   * This method reverse the given list elements but returns the list\_reverseiterator object

Example:

>>> l = [40, 30, 20, 60, 40, 10]

>>> reversed(l)

<list\_reverseiterator object at 0x040EAE10>

---->> To get elements from this list\_reverseiterator object then use list() method.

>>> list(reversed(l)) [10, 40, 60, 20, 30, 40]

1. copy():
   * This function copies the existing list into new variable. Example:

>>> List1 = [10,20,'Python',30,True,'Srinivas',10,3+4j]

>>> x = List1.copy()

>>> print(List1) # [10, 20, 'Python', 30, True, 'Srinivas', 10, (3+4j)]

>>> print(x) # [10, 20, 'Python', 30, True, 'Srinivas', 10, (3+4j)]

* + After copying the given list elements into another list gets new object reference.

Example:

>>> l = [40, 30, 20, 60, 40, 10]

>>> l2 = l.copy()

>>> l2

[40, 30, 20, 60, 40, 10]

>>> id(l) 67976448

>>> id(l2) 63174056

1. clear():
   * This function clears the entire elements from the given list. Example:

>>> List1 = [10,20,'Python',30,True,'Srinivas',10,3+4j]

>>> List1.clear()

>>> print(List1) []

>>> del List1 12.max():

* + This functions finds the maximum value in the given list Example:

>>>List2=[10,20,30,40]

>>> max(List2) 40

1. min():
   * This function finds the minimum value in the given list Example:

>>> List2=[10,20,30,40]

>>> min(List2) 10

1. sort():
   * This function sorts the elements of given list.

>>> lst=[1,9,5,11,2]

>>> lst.sort()

>>> print(lst) [1, 2, 5, 9, 11]

>>> l1=[1,2,5,3,7,4,2,True]

>>> l1.sort()

>>>print(l1) [1, True, 2, 2, 3, 4, 5, 7]

Note: sort() method sorts the elements in given list only but not assign to new variable.

Example:

>>> l = [10,40,60,30,20]

>>> l2 = l.sort()

>>> l2

>>>

>>> l

[10, 20, 30, 40, 60]

Note: by default this function sorts in ascending order, we can also get in descending order by setting True for reverse.

Example:

>>> lst=[1,9,5,11,2]

>>> lst.sort(reverse=True)

>>> print(lst) [11, 9, 5, 2, 1] Example:

>>> l1=[1,2,5,3,7,4,2,True]

>>> l1.sort(reverse=True)

>>>print(l1) [7, 5, 4, 3, 2, 2, 1, True]

del command:

* + This command is used to remove any specific element in the list or to remove entire list object permanently.

Removing specific element:

>>> List1 = [10,20,'Python',30,True,'Srinivas',10,3+4j]

>>> del List1[0]

>>> print(List1) # [20, 'Python', 30, True, 'Srinivas', 10, (3+4j)]

>>> List1 = [10,20,'Python',30,True,'Srinivas',10,3+4j]

>>> del List1[4]

>>> print(List1) [10, 20, 'Python', 30, 'Srinivas', 10, (3+4j)]

Removing entire list object:

Example:

>>> List1 = [10,20,'Python',30,True,'Srinivas',10,3+4j]

>>> del List1 # deleting lst

>>> print(List1) # after deleting

**NameError**: name 'lst' is not defined

Nested List:

* + Python supports Nested lists also, it means a list contains another lists.

>>> List1 = [10,'Python',5.5]

>>> List2 = [20,30,'Srinivas',3+4j]

>>> List3 = [1,True,2,'Durga']

>>> print(List1) [10, 'Python', 5.5]

>>> print(List2) [20, 30, 'Srinivas', (3+4j)]

>>> print(List3) [1, True, 2, 'Durga']

>>> NestList=[List1,List2,List3] #creating a list by using existing lists.

>>> print(NestList)

[[10, 'Python', 5.5], [20, 30, 'Srinivas', (3+4j)], [1, True, 2, Durga']]

>>> type(NestList) <class 'list'>

>>> print(NestList[0]) [10, 'Python', 5.5]

>>> print(NestList[1]) [20, 30, 'Srinivas', (3+4j)]

>>> print(NestList[2]) [1, True, 2, 'Durga']

>>>print(NestList[0][0]) 10

>>>print(NestList[1][0]) 20

>>>print(NestList[2][0]) 1

Converting a String into List :

>>> Str1 = 'Python is very simple and easy language'

>>> print(Str1) # Python is very simple and easy language

>>> type(Str1) # <class 'str'>

>>> List1 = Str1.split()

>>> print(List1) # ['Python', 'is', 'very', 'simple', 'and', 'easy', 'language']

>>> type(List1) # <class 'list'>

Converting a List to String :

>>>List1=['Python', 'is', 'very', 'simple', 'and', 'easy', 'language']

>>> print(List1) # ['Python', 'is', 'very', 'simple', 'and', 'easy', 'language']

>>> type(List1) <class 'list'>

>>> Str2=" ".join(List1)

>>> print(Str2) Python is very simple and easy language

>>> type(Str2) <class 'str'>

The main difference between String and List is mutation.

1. String is immutable whereas List is mutable.
2. Mutable objects can be altered whereas immutable objects can’t be altered.

List Packing :

A list can be created by using a group of variables, it is called list packing

>>> a=10

>>> b=20

>>> c=True

>>> d='Py'

>>> list1=[a,b,c,d]

>>> print(list1) [10, 20, True, 'Py']

>>> type(list1) <class 'list'>

List Unpacking :

1. List unpacking allows to extract list elements automatically.
2. List unpacking is the list of variables on the left has the same number of elements as the length of the list.

>>> lst=[10,20,'Python',True]

>>> print(lst) [10, 20, 'Python', True]

>>> type(lst) <class 'list'>

>>> a,b,c,d=lst #list unpacking

>>> print(a) 10

>>> type(a) <class 'int'>

>>> print(b) 20

>>> type(b) <class 'int'>

>>> print(c) Python

>>> type(c) <class 'str'>

>>> print(d) True

>>> type(d) <class 'bool'>

### SET- DATA Type:

* A set is unordered collection of unique elements.
* Set is commonly used in membership testing, removing duplicates from a sequence, and computing mathematical operations such as intersection, union, difference, and symmetric difference.
* Set will not allow duplicate values.
* Insertion order is not preserved but elements can be sorted
* The major advantage of using a set is as opposed to a list, is that it has a highly optimized method for checking whether a specific element is contained in the set.
* Sets do not support indexing, slicing,
* Sets do not support concatenation and multiplication.
* There are currently two built-in set types,
  1. set,
  2. frozenset.

### Set:

* The set type is mutable means the contents of set can be changed using methods like add(), update() and remove(), discard(), pop() , clear().
* Since it is mutable, it has no hash value and cannot be used as either a dictionary key or as an element of another set.

### Frozenset:

* The frozen sets are the immutable form of the normal sets, i.e., the items of the frozen set cannot be changed and therefore it can be used as a key in the dictionary.
* The elements of the frozen set cannot be changed after the creation. We cannot change or append the content of the frozen sets by using the methods like add() or remove().
* The frozenset() method is used to create the frozenset object. The iterable sequence is passed into this method which is converted into the frozen set as a return type of the method.

Consider the following example to create the frozen set.

Frozenset = frozenset([1,2,3,4,5]) print(type(Frozenset))

print("\nprinting the content of frozen set...") for I in Frozenset:

print(i); Frozenset.add(6)

#gives an error since we cannot change the content of Frozenset after creation

We can create a set in different ways,

1. **Creating an empty set using set() and add elements to that empty set.**

Example:

>>> set1 = set() #creating an empty set with set()

>>> set1.add(10) #adding elements to empty set

>>> set1.add(20) #adding elements to empty set

>>> set1.add(30) #adding elements to empty set

>>> set1.add(10) #adding duplicate value to set

>>> print(set1) {10, 20, 30}

1. Creating a set with elements using set().

Example:

>>> set2=set([1,2,4,'a',2+4j,True]) #creating set with set()

>>> print(set2) {1, 2, 4, (2+4j), 'a'}

>>> type(set2) <class 'set'>

1. Creating a set with curly braces ----->> { }

Example:

>>> set3={1,2,3,4,"Srinivas",True} #creating a set with curly braces

>>> print(set3) {1, 2, 3, 4, 'Srinivas'}

>>> type(set3) <class 'set'>

identify differences

s3 = {1,20,40,True,0,30,False}

>>> s3

{0, 1, 40, 20, 30}

>>>

>>> s3 = {20,False, 40,True,0,1,30}

>>> s3

{False, True, 40, 20, 30}

### Set Functions:

Adding items to the set:

* Python provides the add() method and update() method which can be used to add some particular item to the set. The add() method is used to add a single element whereas the update() method is used to add multiple elements to the set.

add():

* This method is used to add new elements in to existing set. Example:

>>> set1 = {1,2,3,4,5}

>>> print(set1) {1, 2, 3, 4, 5}

>>> set1.add(6) # adding element 6

>>> set1.add(7) # adding element 7

>> print(set1) {1, 2, 3, 4, 5, 6, 7}

Note: we can not add new elements to the frozenset.

>>>fs = frozenset([10,20,30,40])

>>>print(fs) {10,20,30,40}

>>> fs.add(50) #trying to add new element to frozenset. Error: **AttributeError**: 'frozenset' object has no attribute 'add'

update():

* To add more than one item in the set, Python provides the update() method. It accepts iterable object as an argument.

Example:

>>> s1 = set()

>>> s1.update([10,20,30,40])

>>> s1

{40, 10, 20, 30}

>>> s1.update((50,60))

>>> s1 # {10,20,60,50,40,30}

##### Removing items from the set:

remove(element):

* It will remove elements from the set, if that element is not found then it will throw error like **KeyError**

Example:

>>> se1={1,2,3,4,5}

>>> print(se1) # {1, 2, 3, 4, 5}

>>> type(se1) # <class 'set'>

>>> se1.remove(5) # removing element from set

>>> se1.remove(4) # removing element from set

>>> se1.remove(15) # trying to remove element which is not there in set Error: **KeyError**: 15

discard():

* It will remove elements from the set, if that element is not found in the set then it will do nothing. means it will not return any exception here.

Example:

>>> se1={1,2,3,4,5}

>>> print(se1) # {1, 2, 3, 4, 5}

>>> se1.discard(7) # trying to remove element which not there in the set.

>>> se1.discard(20) # trying to remove element which not there in the set.

>>> se1.discard(5) # removing element which is there in the set

>>> print(se1) # {1, 2, 3, 4}

Q. What is the difference between remove() and discard() ? remove():

* It will remove element from the given set if it is a member of given set object. If we take the element which is not present in the set then it will throws error like **KeyError**.

discard():

* It will remove element from the given set if it is a member of given set object. If we take the element which is not present in the set then it will do nothing, means it will not throw error.

pop():

* We can also use the pop() method to remove the item. Generally, the pop() method will always remove the last item but the set is unordered, we can't determine which element will be popped out from set.

Example:

>>> s1 = {90,40, 10, 20, 30}

>>> s1.pop()

40

>>> s1.pop() 10

Note:

* If given set object not contains any elements then pop() method returns "**keyError**" when trying to delete element from set object

Example:

>>> s2 = set()

>>> s2.pop()

**KeyError**: 'pop from an empty set'

Assignment operator

* By using assignment operator, if we assigning given set object in to another object then both can share the same memory address.

Example:

>>> set1 = {1, 2, 3, 4, 5}

>>> set2 = set1

>>> set1

{1, 2, 3, 4, 5}

>>> set2

{1, 2, 3, 4, 5}

>>> id(set1) 2693878076136

>>> id(set3) 2693878076136

copy():

* This function copies the elements of one set to another new set and also it creates new memory value.
* copy() method always creates new memory for new set object. so both memories are different but values are same.

Example:

>>> set1={1,2,3,4,5}

>>> set2=set1.copy() #copying se1 elements to se2

>>> set1 {1, 2, 3, 4, 5}

>>> set2 {1, 2, 3, 4, 5} # id values different.

>>> id(set1) 2693878076136

>>> id(set2) 2693878076360

clear():

* By using clear() function we can clear or remove all elements from the given set object.

Example:

>>> se1={1,2,3,4,5}

>>> print(se1) {1, 2, 3, 4, 5}

>>> type(se1) <class 'set'>

>>> se1.clear() #clearing the se1, so se1 will become empty set.

>>> print(se1) set()

##### Python Set Operations:

* Set can be performed mathematical operation such as union, intersection, difference, and symmetric difference. Python provides the facility to carry out these operations with operators or methods. We describe these operations as follows.

isdisjoint():

* This function returns True if both are "empty sets" or if both sets "contains non-matching" elements.
* if atleast one elemet matching also returns False value. Example:

>>> se1 = set()

>>> se2 = set()

>>> se1.isdisjoint(se2) True

>>> se1=set(5)

>>> se2={1,2,3}

>>> se1.isdisjoint(se2) True

>>> se1={1,2,3}

>>> se2={1,2,3,4}

>>> se1.isdisjoint(se2) False issubset():

* x.issubset(y) returns True, if x is a subset of y.
* " <= " is an abbreviation for "Subset of". For example:

>>> se1={1,2,3,4,5}

>>> se2={1,2,3}

>>> se2.issubset(se1) True

>>> se1.issubset(se2) False Or

>>> se2 <= se1 True

>>> se1 <= se2 False

issuperset()

* x.issuperset(y) returns True, if x is a superset of y.
* " >= " is an abbreviation for "issuperset of" Example:

>>> se1={1,2,3,4,5}

>>> se2={1,2,3}

>>> se2.issuperset(se1) False

>>> se1.issuperset(se2) True

>>> se2 >= se1 False

>>> se1 >= se2 True

Membership:

* We can also check the elements whether they belong to set or not, Example:

>> se1={1,2,3,"Python",3+5j,8}

>>> 4 in se1 False

>>> 1 in se1 True

>>> "Python" in se1 True

>>> 10 not in se1 True

>>> "Srinivas" not in se1 True

union():

* It returns the union of two sets, that means it returns all the values from both sets except duplicate values.
* The same result we can get by using ‘|’ between two sets Syntax: <First\_Set>.union(<Second\_Set>) or

<First\_Set> | <Second\_Set> Example:

>>> se1={1,2,3,4,5}

>>> se2={1,2,3,6,7}

>>> se1.union(se2) {1, 2, 3, 4, 5, 6, 7} or

>>> se1|se2 {1, 2, 3, 4, 5, 6, 7}

Or

>>> se2.union(se1) {1, 2, 3, 4, 5, 6, 7}

>>> se2|se1 {1, 2, 3, 4, 5, 6, 7}

intersection():

* It returns an intersection elements of two sets, that means it returns only common elements from both sets.
* That same operation we can get by sing ‘&’ operator. Syntax: <First\_Set>.intersection(<Second\_Set>) or

<First\_Set> & <Second\_Set> Example:

>>> se1={1,2,3,4,5}

>>> se2={1,2,3,6,7}

>>> se1.intersection(se2) {1, 2, 3} or

>>> se1&se2 {1, 2, 3}

Or

>>> se2.intersection(se1) {1, 2, 3}

>>> se2&se1 {1, 2, 3}

diffferenece():

* It returns all elements from first set which are not there in the second set. Syntax: <First\_set>.difference(<Secnd\_Set>) or

<First\_Set> - <Second\_Set> Example:

>>> se1={1,2,3,4,5}

>>> se2={1,2,3,6,7}

>>> se1.difference(se2) {4, 5} or

>>> se1-se2 {4, 5}

Or

>>> se2.difference(se1) {6, 7} or

>>> se2-se1 {6, 7}

intersection\_update():

* The intersection\_update() method removes the items from the original set that are not present in both the sets (all the sets if more than one are specified).
* The intersection\_update() method is different from the intersection() method since it modifies the original set by removing the unwanted items, on the other hand, the intersection() method returns a new set.

Syntax: <First\_Set>.intersection\_update(<Second\_Set>) Example:

>>> se1={1,2,3,4,5}

>>> se2={1,2,3,6,7}

>>> se1.intersection\_update(se2)

>>> print(se1) {1, 2, 3}

>>> print(se2) {1, 2, 3, 6, 7}

Or

>>> se1={1,2,3,4,5}

>>> se2={1,2,3,6,7}

>>> se2.intersection\_update(se1)

>>> print(se1) {1, 2, 3, 4, 5}

>>> print(se2) {1, 2, 3}

differenece\_update():

* The result of difference between two sets will in First\_Set. Syntax: <First\_Set>.difference\_update(<Second\_Set>) Example:

>>> se1={1,2,3,4,5}

>>> se2={1,2,3,6,7}

>>> se1.difference\_update(se2)

>>> print(se1) {4, 5}

>>> print(se2) {1, 2, 3, 6, 7}

Or

>>> se1={1,2,3,4,5}

>>> se2={1,2,3,6,7}

>>> se2.difference\_update(se1)

>>> print(se1) {1, 2, 3, 4, 5}

>>> print(se2) {6, 7}

symmetric\_difference():

* The symmetric difference of two sets is calculated by ^ operator or symmetric\_difference() method. Symmetric difference of sets, it removes that element which is present in both sets.
* It returns unmatching elements from both sets. Syntax: <First\_Set>.symmetric\_difference(<Second\_Set>) Example:

>>> set1={1,2,3,4,5}

>>> set2={1,2,3,6,7}

>>> set1.symmetric\_difference(set2) {4, 5, 6, 7} or

>>> set1 ^ set2 {4, 5, 6, 7}

symmetric\_difference\_update():

* it will store the unmatching elements from both sets into First\_Set. Syntax: <First\_Set>.symmetric\_difference\_update(<Secon\_Set>) Example:

>>> set1 = {1,2,3,4,5}

>>> set2 = {1,2,3,6,7}

>>> set1.symmetric\_difference\_update(set2)

>>> print(set1) {4, 5, 6, 7}

>>> print(set2) {1, 2, 3, 6, 7}

Or

>>> set1 ={1,2,3,4,5}

>>> set2 = {1,2,3,6,7}

>>> set2.symmetric\_difference\_update(set1)

>>> print(set1) {1, 2, 3, 4, 5}

>>> print(set2) {4, 5, 6, 7}

set ():

Example:

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

>>> s {10, 20, 30}

>>> s.add(40) # add new element into set

>>> s {40, 10, 20, 30}

>>> s.add(40,50) # TypeError: add() takes exactly one argument (2 given)

>>> s.add([80,40,50]) # TypeError: unhashable type: 'list'

>>> s.add((80,40,50))

>>> s # {40, 10, (80, 40, 50), 20, 30}

Q. Is set object allowed Mutable type data ?

Set object not allowed mutable type objects data in side set object.

It means, list , set and dictionary type data not allowed in set object either directly or indirectly.

For example:

>>> s = {1 , 2.0 , [1 , 2]}

TypeError: unhashable type: 'list'

>>> s = {1,2.0, {1,2}}

TypeError: unhashable type: 'set'

>>> s = {1,2.0, {1:2}}

TypeError: unhashable type: 'dict'

>>> s = { 1 , 2.0 , 'python' , (1 , 2 , [ 1 , 2 ] ) }

TypeError: unhashable type: 'list'

Q. Is set object allowed Immutable type data ?

Yes, set allowed only immutable type objects data inside set object.

It means, numbers , string and tuple type data allowed in set object either directly or indirectly.

For example:

>>> s1 = { 1 , 2 , ( 1 , 2 ) , 'python' }

>>> s1

{ 1 , 2 , ( 1 , 2 ) , 'python' }

##### Python List Interview Questions

**What is a Python list?**

1. **How do you create an empty list in Python?**
2. **How do you create a list with initial values in Python?**
3. **How do you access elements in a list?**
4. **How do you add an element to the end of a list?**
5. **How do you add an element at a specific position in a list?**
6. **How do you remove an element from a list?**
7. **How do you check if an element is in a list?**
8. **How do you find the length of a list?**
9. **How do you concatenate two lists?**
10. **How do you iterate over a list?**
11. **How do you reverse a list?**
12. **How do you sort a list?**
13. **How do you clear a list elements?**
14. **What are differences between append() and extend() methods in list?**

**What are differences between remove() and pop() methods in list?**

**16.**

**What are differences between index() and insert() methods in list?**

**17.**

**Is list a Mutable or Immutable object ? Why ?**

**18.**

* A list in Python is a collection of elements which is ordered and changeable. It allows duplicate members.

**2. How do you create an empty list in Python?**

**1. What is a Python list?**

**==================== ================== ===================**

**How to find the occurences of given value in a list object ?**

**20.**

**How to repeate a list object multiple times ?**

**19.**

my\_list = []

**How do you create a list with initial values in Python?**

my\_list = [1, 2, 3, 4, 5]

**4. How do you access elements in a list?**

* + an access elements in a list using indexing. Indexing starts from 0.
  + **For example**,

# accesses the first element.

**my\_list[0]**

**5.**

|  |  |  |
| --- | --- | --- |
| **How do you add an element to the end of a list?** | | |
| * u can use the | **append()** | method. |

my\_list.append(6)

|  |  |  |
| --- | --- | --- |
| **6. How do you add an element at a specific position in a list?** | | |
| * u can use the | **insert()** | method. |

my\_list.insert(2, 7) # Insert 7 at index 2

* + ou can use the remove.

**remove()**

**7. How do you remove an element from a list?**

method if you know the value of the element you want to

my\_list.remove(3) # Removes the first occurrence of 3

* + You can use the

method if you want to remove an element by its index.

my\_list.pop(2) # Removes the element at index 2

**pop()**

|  |  |  |
| --- | --- | --- |
| **8. How do you check if an element is in a list?** | | |
| * can use the | **in** | keyword. |

if 5 in my\_list:

print("5 is in the list")

|  |  |  |
| --- | --- | --- |
| **9. How do you find the length of a list?** | | |
| * u can use the | **len()** | function. |

length = len(my\_list)

|  |  |  |
| --- | --- | --- |
| **10. How do you concatenate two lists?** | | |
| * can use the | **+** | operator. |

new\_list = my\_list + [8, 9, 10]

**11. How do you iterate over a list?**

* + u can use a

loop.

for item in my\_list: print(`item)

**for**

method.

|  |  |  |
| --- | --- | --- |
| **12. How do you reverse a list?** | | |
| * can use the | **reverse()** |  |

my\_list.reverse()

|  |  |  |
| --- | --- | --- |
| **13. How do you sort a list?** | | |
| * the | **sort()** | method. |

my\_list.sort()

###### Python Set Datatype Interview Questions

**1. What is a set in Python?**

**2. How do you create an empty set in Python?**

**3. How are sets different from lists and tuples?**

**4. How do you add elements to a set?**

**5. How do you add elements to a set?**

**6. How do you perform set operations like union, intersection, and difference in Python?**

**7. How do you check if a set is a subset or superset of another set?**

**8. Can you sort a set in Python?**

**9. What are the differences between remove() and discard() methods in Set ?**

**10. What are the differences between add() and update() method in Set ?**

**===================== =================== =====================**

**What is a set in Python?**

* A set is an unordered collection of unique elements in Python. It is defined by enclosing a comma-separated list of elements within curly braces **{}**.

**How do you create an empty set in Python?**

* An empty set in Python can be created using the using curly braces **{}**.

**set()**

**For example: empty\_set = set()**

constructor method only but not

**# Note: empty curly braces are reserved for “dict” type only. empty\_set = {}**

**How are sets different from lists and tuples?**

* Sets are unordered collections of unique elements, while lists and tuples are ordered collections that can contain duplicate elements. Additionally, sets are mutable, meaning you can add or remove elements, whereas tuples are immutable, and lists are mutable.

**How do you add elements to a set?**

* You can add elements to a set using the

**add()**

**my\_set = {1, 2, 3} my\_set.add(4)**

method. For example:

**How do you remove elements from a set?**

* You can remove elements from a set using the

method or the

method.

The difference is that will raise a KeyError if the element does not exist, while

**remove()**

**discard()**

**remove()**

**discard()** will not raise any error. For example:

**my\_set = {1, 2, 3} my\_set.remove(3) # or my\_set.discard(3)**

**How do you perform set operations like union, intersection, and difference in Python?**

* Python provides operators and methods for set operations. For example:

**set1 = {1, 2, 3}**

**set2 = {3, 4, 5}**

**# Union**

**union\_set = set1 | set2 # Using | operator**

**union\_set = set1.union(set2) # Using union() method**

**# Intersection**

**intersection\_set = set1 & set2 # Using & operator**

**intersection\_set = set1.intersection(set2) # Using intersection() method**

**# Difference**

**difference\_set = set1 - set2 # Using - operator**

**difference\_set = set1.difference(set2) # Using difference() method**

**How do you check if a set is a subset or superset of another set?**

**issuperset()**

**issubset()**

* You can use the

and

methods to check for subset and superset

relationships, respectively. For example:

**set1 = {1, 2}**

**set2 = {1, 2, 3}**

**print(set1.issubset(set2)) # True print(set2.issuperset(set1)) # True**

**Can you sort a set in Python?**

* No, sets are inherently unordered collections, so they cannot be sorted. If you need a sorted collection, you can convert the set to a sorted list using the **sorted()** function. For example:

**my\_set = {3, 1, 2} sorted\_list = sorted(my\_set)**



**Python Programming**

## Module-1: Python Introduction

1. **Who should learn Python?**
2. **Advantages of Learning Python.**
3. **Why Python demand in current Industry**
4. **Python History**
5. **Python Features**
6. **Python Installation and PATH setting**
7. **First Python Program Development**

## Module-2: Python Concepts

1. **Python Keywords and Comments Concept**
2. **Use of Quotes and different types**
3. **Python Indentation Concept**
4. **What is an identifier? What are the rules of an identifier?**
5. **Python Variables and different ways to create variables**
6. **Practical examples on variables, Identifiers, keywords**
7. **How to reading input data from user?**
8. **Working with input () function in Python?**
9. **Python Datatypes and Datatype Conversions.**
10. **Type Casting Functions and Examples?**
11. **Python Operators Concept**

## Module-3 : Variables & Data Types

1. **Python Data Types and Data Structures**
2. **Python String Data Structure**
3. **Python Tuple Data Structure**
4. **Python List Data Structure**
5. **Python Set Data Structure**
6. **Python Dictionary Data Structure**
7. **Python Data Packing and Unpacking Concept**
8. **Python data type Comprehensions concept**



## Module-4 : Control Statements

1. **Python Control Statements**
2. **Python Conditional Statements**
3. **Python Looping or Iterative Statements**
4. **Python Transfer or Jump**
5. **Working with enumerate and format functions**

## Module-5 : Functions + Modules + Packages

1. **Python Functions Concept.**
2. **Types of Arguments in Functions.**
3. **Local Variable Scope**
4. **Global Variable Scope**
5. **Scope Conversion**
6. **Python Lambda Expressions**
7. **Working with filter() , map() and reduce() functions.**
8. **Python Modules Concept and Types of Modules?**
9. **How many ways can we import a module in Python?**
10. **Working with modules**
11. **Python Packages**

## Module – 6 : Files + Exceptions

1. **Python Files Handling concept.**
2. **Performing CURD Operations in Files?**
3. **Working with os module for handling files.**
4. **Interview Questions on Files Handling concept.**
5. **Python Exception Handling concept.**
6. **Runtime Exceptions**
7. **Userdefined Exceptions**



## Module-7 : OOPS

1. **What is OOPS**
2. **Class + Object + Method + Constructor + Variables**
3. **OOPS Principles**
4. **Working with Inheritance and Its types ?**
5. **Working with Polymorphism and Its types?**
6. **Working with Encapsulation and Its types?**
7. **Working with Abstraction and Examples?**

## Module-8 : Python with Database Comm..

1. **Python with Database Connection Concept. (MySQL DB, Mongo DB, Oracle DB)**

## Module-9:

1. **Python Interview Questions & Answers**