PYTHON PROGRAMMING LANGUAGE

Python is a high-level, interpreted, and object-oriented programming language known for its simplicity and readability. It was created by Guido van Rossum and first released in 1991. Python's design philosophy emphasizes code readability, using significant indentation to define code blocks, which makes it easier to understand and write.

Why python for beginners?

- Easy to learn(shorter code than java , c…).
- Open source(no need for license for using python, simply download it and use it)
- oop(object-oriented programming-which supports,inheritance,abstraction,polymorphism,encapsulation,exception handling , reusability of code is more, code is highly secured, modularity concept is proposed)
- procedural functional.
- Portable(write once run anywhere),GUI programming.
- Employed in broad range of applications rising demand for python programmers.

Python Interfaces and Installations:

IDLE , COLAB,JYUPTER

Features of python:

- High-level interpreted interactive language.
- Extensive standard libraries.
- Cross platform compatibility.
- Portable and extensible.
- Supports databases and gui programming
- Scalable and dynamic semantics
- Automatic garbage collection

Tokens of python:

Types of comments in python:

PYTHON PROGRAMMING LANGUAGE

There are two types of comments in python:

- Single-line comment:Represented with hashtag.

  Ex:#single-line comment

- Multi-line comment: represented using front slash and an astrick.

  Ex:/*this

  Is multi-line comment*/

The comments improve readability of the program. They use to document software requirements and associated information properly.

Types of Data in python:

1. Numeric data(integer,real ,float data)
2. String data(alphanumeric data)
3. Boolean data
4. None data(missing data)

Numeric data: The numeric data type in Python represents the data that has a numeric value. A numeric value can be an integer, a floating number, or even a complex number.

- Integers – This value is represented by int class. It contains positive or negative whole numbers (without fractions or decimals). In Python, there is no limit to how long an integer value can be.
- Float – This value is represented by the float class. It is a real number with a floating-point representation. It is specified by a decimal point. Optionally, the character e or E followed by a positive or negative integer may be appended to specify scientific notation.
- Complex Numbers – A complex number is represented by a complex class. It is specified as *(real part) + (imaginary part)j* . For example – 2+3j

Ex:

```
a = 5
print(type(a))
b = 5.0
print(type(b))
c = 2 + 4j
print(type(c))
```

output:

```
<class 'int'>
<class 'float'>
<class 'complex'>
```

2. Sequence Data Types in Python

The sequence Data Type in Python is the ordered collection of similar or different Python data types. Sequences allow storing of multiple values in an organized and efficient fashion.

- **String Data Type**

Python Strings are arrays of bytes representing Unicode characters. In Python, there is no character data type Python, a character is a string of length one. It is represented by str class.

Strings in Python can be created using single quotes, double quotes or even triple quotes. We can access individual characters of a String using index.

Ex:

```
s = 'Welcome to the Amar World'
print(s)
# check data type
print(type(s))
# access string with index
print(s[1])
print(s[2])
print(s[-1])
```

Output:

```
Welcome to the Amar World
<class 'str'>
e
l
d
```

- **List Data Type**

Lists are just like arrays, declared in other languages which is an ordered collection of data. It is very flexible as the items in a list do not need to be of the same type.

Creating a List in Python

Lists in Python can be created by just placing the sequence inside the square brackets[].

Ex: # Empty list

```
a = []

# list with int values

a = [1, 2, 3]

print(a)

# list with mixed int and string

b = ["Amar", "For", "Amar", 4, 5]

print(b)
```

Output:

[1, 2, 3]

[Amar, 'For', Amar, 4, 5]

Access List Items

In order to access the list items refer to the index number. In Python, negative sequence indexes represent positions from the end of the array. Instead of having to compute the offset as in List[len(List)-3], it is enough to just write List[-3]. Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second-last item, etc.

```
a = ["Ayan", "For", "Ayan"]

print("Accessing element from the list")

print(a[0])

print(a[2])

print("Accessing element using negative indexing")

print(a[-1])

print(a[-3])
```

Output

Accessing element from the list

Ayan

Ayan

Accessing element using negative indexing

Ayan

Ayan

- **Tuple Data Type**

Just like a list, a tuple is also an ordered collection of Python objects. The only difference between a tuple and a list is that tuples are immutable. Tuples cannot be modified after it is created.

Creating a Tuple in Python

In Python Data Types, tuples are created by placing a sequence of values separated by a 'comma' with or without the use of parentheses for grouping the data sequence. Tuples can contain any number of elements and of any datatype (like strings, integers, lists, etc.

```
# initiate empty tuple

tup1 = ()

tup2 = (Amit, 'For')

print("\nTuple with the use of String: ", t2)
```

Output

Tuple with the use of String:
(Amit, 'For')

Note *– The creation of a Python tuple without the use of parentheses is known as Tuple Packing.*

Access Tuple Items

In order to access the tuple items refer to the index number. Use the index operator [ ] to access an item in a tuple.

```
tup1 = tuple([1, 2, 3, 4, 5])

# access tuple items

print(tup1[0])

print(tup1[-1])

print(tup1[-3])
```

Output

1

5

3

- **Boolean Data Type**

Python Data type with one of the two built-in values, True or False. Boolean objects that are equal to True are truthy (true), and those equal to False are falsy (false). However non-Boolean objects can be evaluated in a Boolean context as well and determined to be true or false. It is denoted by the class bool.

Example: The first two lines will print the type of the boolean values True and False, which is **<class 'bool'>.** The third line will cause an error, because true is not a valid keyword in Python. Python is case-sensitive, which means it distinguishes between uppercase and lowercase letters.

```
print(type(True))
print(type(False))
```

Output:

```
<class 'bool'>
<class 'bool'>
```

- **None data(missing data)**

In Python, missing data is often represented by None in regular Python objects and NaN (Not a Number) in numerical computations using libraries like NumPy and Pandas.

Understanding None in Python

- None is a special keyword representing the absence of a value.
- It is **not** the same as 0, False, or an empty string ("").
- None is an instance of the NoneType class.

Example:

```
x = None
print(x)  # Output: None
print(type(x))  # Output: <class 'NoneType'>
```

Representation of raw data: binary, octal, hexadecimal, decimal.

#Declaration of binary data:

```
b=0b1010
c=0o124
h=0x12D
a=20
print(b)
print(c)
print(h)
print(a)
```

Output:10 84 301 20

Input& Output functions:

Input function:

A=20#this is a static declaration of a variable value.

Dynamically input can be taken from the user using input function.

Ex:

```
name = input("Enter your name")
print(name)
```

Output:

Enter your name: Tejaswini

Tejaswini

Note: By default the input is considered as a string if you pass anything.

Ex:

#perform arithmetic addition by taking input values from the user or at runtime.

```
a=input("Enter a value:")
b=input("Enter b value.")
print("sum =",a+b)
```

Output:

Enter a value:10

Enter b value:20

Sum=1020

Note :

Here the a and b values arer considered as the strings by the input function and by using the concatenation method we get the output as 1020.

#To avoid this explicit type cast is required to drawback of input function:

```
a=int(input("Enter your  a value:"))
b=int(input("Enter your b value:"))
print("sum=",a+b)
```

Output:

Enter a value:10

Enter b value:20

Sum=30

Output Functions in Python

In Python, we use the print() function as the primary way to output data. Below are different ways to use the print() function effectively.

PYTHON PROGRAMMING LANGUAGE

Python is a high-level, interpreted, and object-oriented programming language known for its simplicity and readability. It was created by Guido van Rossum and first released in 1991. Python's design philosophy emphasizes code readability, using significant indentation to define code blocks, which makes it easier to understand and write.

Why python for beginners?

- Easy to learn(shorter code than java , c…).
- Open source(no need for license for using python, simply download it and use it)
- oop(object-oriented programming-which supports,inheritance,abstraction,polymorphism,encapsulation,exception handling , reusability of code is more, code is highly secured, modularity concept is proposed)
- procedural functional.
- Portable(write once run anywhere),GUI programming.
- Employed in broad range of applications rising demand for python programmers.

Python Interfaces and Installations:

IDLE , COLAB,JYUPTER

Features of python:

- High-level interpreted interactive language.
- Extensive standard libraries.
- Cross platform compatibility.
- Portable and extensible.
- Supports databases and gui programming
- Scalable and dynamic semantics
- Automatic garbage collection

Tokens of python:

Types of comments in python:

PYTHON PROGRAMMING LANGUAGE

There are two types of comments in python:

- Single-line comment:Represented with hashtag.

  Ex:#single-line comment

- Multi-line comment: represented using front slash and an astrick.

  Ex:/*this

  Is multi-line comment*/

The comments improve readability of the program. They use to document software requirements and associated information properly.

Types of Data in python:

5. Numeric data(integer,real ,float data)
6. String data(alphanumeric data)
7. Boolean data
8. None data(missing data)

Numeric data: The numeric data type in Python represents the data that has a numeric value. A numeric value can be an integer, a floating number, or even a complex number.

- Integers – This value is represented by int class. It contains positive or negative whole numbers (without fractions or decimals). In Python, there is no limit to how long an integer value can be.
- Float – This value is represented by the float class. It is a real number with a floating-point representation. It is specified by a decimal point. Optionally, the character e or E followed by a positive or negative integer may be appended to specify scientific notation.
- Complex Numbers – A complex number is represented by a complex class. It is specified as *(real part) + (imaginary part)j* . For example – 2+3j

Ex:

```
a = 5
print(type(a))
b = 5.0
print(type(b))
c = 2 + 4j
print(type(c))
```

output:

```
<class 'int'>
<class 'float'>
<class 'complex'>
```

2. Sequence Data Types in Python

The sequence Data Type in Python is the ordered collection of similar or different Python data types. Sequences allow storing of multiple values in an organized and efficient fashion.

- **String Data Type**

Python Strings are arrays of bytes representing Unicode characters. In Python, there is no character data type Python, a character is a string of length one. It is represented by str class.

Strings in Python can be created using single quotes, double quotes or even triple quotes. We can access individual characters of a String using index.

Ex:

```
s = 'Welcome to the Amar World'
print(s)
# check data type
print(type(s))
# access string with index
print(s[1])
print(s[2])
print(s[-1])
```

Output:

```
Welcome to the Amar World
<class 'str'>
e
l
d
```

- **List Data Type**

Lists are just like arrays, declared in other languages which is an ordered collection of data. It is very flexible as the items in a list do not need to be of the same type.

Creating a List in Python

Lists in Python can be created by just placing the sequence inside the square brackets[].

Ex: # Empty list

```
a = []

# list with int values

a = [1, 2, 3]

print(a)

# list with mixed int and string

b = ["Amar", "For", "Amar", 4, 5]

print(b)
```

Output:

[1, 2, 3]

[Amar, 'For', Amar, 4, 5]

Access List Items

In order to access the list items refer to the index number. In Python, negative sequence indexes represent positions from the end of the array. Instead of having to compute the offset as in List[len(List)-3], it is enough to just write List[-3]. Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second-last item, etc.

```
a = ["Ayan", "For", "Ayan"]

print("Accessing element from the list")

print(a[0])

print(a[2])

print("Accessing element using negative indexing")

print(a[-1])

print(a[-3])
```

Output

Accessing element from the list

Ayan

Ayan

Accessing element using negative indexing

Ayan

Ayan

- **Tuple Data Type**

Just like a list, a tuple is also an ordered collection of Python objects. The only difference between a tuple and a list is that tuples are immutable. Tuples cannot be modified after it is created.

Creating a Tuple in Python

In Python Data Types, tuples are created by placing a sequence of values separated by a 'comma' with or without the use of parentheses for grouping the data sequence. Tuples can contain any number of elements and of any datatype (like strings, integers, lists, etc.

```
# initiate empty tuple

tup1 = ()

tup2 = (Amit, 'For')

print("\nTuple with the use of String: ", t2)
```

Output

Tuple with the use of String:
(Amit, 'For')

Note *– The creation of a Python tuple without the use of parentheses is known as Tuple Packing.*

Access Tuple Items

In order to access the tuple items refer to the index number. Use the index operator [ ] to access an item in a tuple.

```
tup1 = tuple([1, 2, 3, 4, 5])

# access tuple items

print(tup1[0])

print(tup1[-1])

print(tup1[-3])
```

Output

1

5

3

- **Boolean Data Type**

Python Data type with one of the two built-in values, True or False. Boolean objects that are equal to True are truthy (true), and those equal to False are falsy (false). However non-Boolean objects can be evaluated in a Boolean context as well and determined to be true or false. It is denoted by the class bool.

Example: The first two lines will print the type of the boolean values True and False, which is **<class 'bool'>.** The third line will cause an error, because true is not a valid keyword in Python. Python is case-sensitive, which means it distinguishes between uppercase and lowercase letters.

```
print(type(True))
print(type(False))
```

Output:

```
<class 'bool'>
<class 'bool'>
```

- **None data(missing data)**

In Python, missing data is often represented by None in regular Python objects and NaN (Not a Number) in numerical computations using libraries like NumPy and Pandas.

Understanding None in Python

- None is a special keyword representing the absence of a value.
- It is **not** the same as 0, False, or an empty string ("").
- None is an instance of the NoneType class.

Example:

```
x = None
print(x)  # Output: None
print(type(x))  # Output: <class 'NoneType'>
```

Representation of raw data: binary, octal, hexadecimal, decimal.

#Declaration of binary data:

```
b=0b1010
c=0o124
h=0x12D
a=20
print(b)
print(c)
print(h)
print(a)
```

Output:10 84 301 20

Input& Output functions:

Input function:

A=20#this is a static declaration of a variable value.

Dynamically input can be taken from the user using input function.

Ex:

```
name = input("Enter your name")
print(name)
```

Output:

Enter your name: Tejaswini

Tejaswini

Note: By default the input is considered as a string if you pass anything.

Ex:

#perform arithematic addition by taking input values from the user or at runtime.

```
a=input("Enter a value:")
b=input("Enter b value.")
print("sum =",a+b)
```

Output:

Enter a value:10

Enter b value:20

Sum=1020

Note :

Here the a and b values arer considered as the strings by the input function and by using the concatenation method we get the output as 1020.

#To avoid this explicit type cast is required to drawback of input function:

```
a=int(input("Enter your  a value:"))
b=int(input("Enter your b value:"))
print("sum=",a+b)
```

Output:

Enter a value:10

Enter b value:20

Sum=30

Output Functions in Python

In Python, we use the print() function as the primary way to output data. Below are different ways to use the print() function effectively.

PYTHON PROGRAMMING LANGUAGE

Python is a high-level, interpreted, and object-oriented programming language known for its simplicity and readability. It was created by Guido van Rossum and first released in 1991. Python's design philosophy emphasizes code readability, using significant indentation to define code blocks, which makes it easier to understand and write.

Why python for beginners?

- Easy to learn(shorter code than java , c…).
- Open source(no need for license for using python, simply download it and use it)
- oop(object-oriented programming-which supports,inheritance,abstraction,polymorphism,encapsulation,exception handling , reusability of code is more, code is highly secured, modularity concept is proposed)
- procedural functional.
- Portable(write once run anywhere),GUI programming.
- Employed in broad range of applications rising demand for python programmers.

Python Interfaces and Installations:

IDLE , COLAB,JYUPTER

Features of python:

- High-level interpreted interactive language.
- Extensive standard libraries.
- Cross platform compatibility.
- Portable and extensible.
- Supports databases and gui programming
- Scalable and dynamic semantics
- Automatic garbage collection

Tokens of python:

Types of comments in python:

PYTHON PROGRAMMING LANGUAGE

There are two types of comments in python:

- Single-line comment:Represented with hashtag.

  Ex:#single-line comment

- Multi-line comment: represented using front slash and an astrick.

  Ex:/*this

  Is multi-line comment*/

The comments improve readability of the program. They use to document software requirements and associated information properly.

Types of Data in python:

9. Numeric data(integer,real ,float data)
10. String data(alphanumeric data)
11. Boolean data
12. None data(missing data)

Numeric data: The numeric data type in Python represents the data that has a numeric value. A numeric value can be an integer, a floating number, or even a complex number.

- Integers – This value is represented by int class. It contains positive or negative whole numbers (without fractions or decimals). In Python, there is no limit to how long an integer value can be.
- Float – This value is represented by the float class. It is a real number with a floating-point representation. It is specified by a decimal point. Optionally, the character e or E followed by a positive or negative integer may be appended to specify scientific notation.
- Complex Numbers – A complex number is represented by a complex class. It is specified as *(real part) + (imaginary part)j* . For example – 2+3j

Ex:

```
a = 5
print(type(a))
b = 5.0
print(type(b))
c = 2 + 4j
print(type(c))
```

output:

```
<class 'int'>
<class 'float'>
<class 'complex'>
```

2. Sequence Data Types in Python

The sequence Data Type in Python is the ordered collection of similar or different Python data types. Sequences allow storing of multiple values in an organized and efficient fashion.

- **String Data Type**

Python Strings are arrays of bytes representing Unicode characters. In Python, there is no character data type Python, a character is a string of length one. It is represented by str class.

Strings in Python can be created using single quotes, double quotes or even triple quotes. We can access individual characters of a String using index.

Ex:

```
s = 'Welcome to the Amar World'
print(s)
# check data type
print(type(s))
# access string with index
print(s[1])
print(s[2])
print(s[-1])
```

Output:

```
Welcome to the Amar World
<class 'str'>
e
l
d
```

- **List Data Type**

Lists are just like arrays, declared in other languages which is an ordered collection of data. It is very flexible as the items in a list do not need to be of the same type.

Creating a List in Python

Lists in Python can be created by just placing the sequence inside the square brackets[].

Ex: # Empty list

```
a = []

# list with int values

a = [1, 2, 3]

print(a)

# list with mixed int and string

b = ["Amar", "For", "Amar", 4, 5]

print(b)
```

Output:

[1, 2, 3]

[Amar, 'For', Amar, 4, 5]

Access List Items

In order to access the list items refer to the index number. In Python, negative sequence indexes represent positions from the end of the array. Instead of having to compute the offset as in List[len(List)-3], it is enough to just write List[-3]. Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second-last item, etc.

```
a = ["Ayan", "For", "Ayan"]

print("Accessing element from the list")

print(a[0])

print(a[2])

print("Accessing element using negative indexing")

print(a[-1])

print(a[-3])
```

Output

Accessing element from the list

Ayan

Ayan

Accessing element using negative indexing

Ayan

Ayan

- **Tuple Data Type**

Just like a list, a tuple is also an ordered collection of Python objects. The only difference between a tuple and a list is that tuples are immutable. Tuples cannot be modified after it is created.

Creating a Tuple in Python

In Python Data Types, tuples are created by placing a sequence of values separated by a 'comma' with or without the use of parentheses for grouping the data sequence. Tuples can contain any number of elements and of any datatype (like strings, integers, lists, etc.

```
# initiate empty tuple

tup1 = ()

tup2 = (Amit, 'For')

print("\nTuple with the use of String: ", t2)
```

Output

Tuple with the use of String:
(Amit, 'For')

Note *– The creation of a Python tuple without the use of parentheses is known as Tuple Packing.*

Access Tuple Items

In order to access the tuple items refer to the index number. Use the index operator [ ] to access an item in a tuple.

```
tup1 = tuple([1, 2, 3, 4, 5])

# access tuple items

print(tup1[0])

print(tup1[-1])

print(tup1[-3])
```

Output

1

5

3

- **Boolean Data Type**

Python Data type with one of the two built-in values, True or False. Boolean objects that are equal to True are truthy (true), and those equal to False are falsy (false). However non-Boolean objects can be evaluated in a Boolean context as well and determined to be true or false. It is denoted by the class bool.

Example: The first two lines will print the type of the boolean values True and False, which is **<class 'bool'>.** The third line will cause an error, because true is not a valid keyword in Python. Python is case-sensitive, which means it distinguishes between uppercase and lowercase letters.

```
print(type(True))
print(type(False))
```

Output:

```
<class 'bool'>
<class 'bool'>
```

- **None data(missing data)**

In Python, missing data is often represented by None in regular Python objects and NaN (Not a Number) in numerical computations using libraries like NumPy and Pandas.

Understanding None in Python

- None is a special keyword representing the absence of a value.
- It is **not** the same as 0, False, or an empty string ("").
- None is an instance of the NoneType class.

Example:

```
x = None
print(x)  # Output: None
print(type(x))  # Output: <class 'NoneType'>
```

Representation of raw data: binary, octal, hexadecimal, decimal.

#Declaration of binary data:

```
b=0b1010
c=0o124
h=0x12D
a=20
print(b)
print(c)
print(h)
print(a)
```

Output:10 84 301 20

Input& Output functions:

Input function:

A=20#this is a static declaration of a variable value.

Dynamically input can be taken from the user using input function.

Ex:

```
name = input("Enter your name")
print(name)
```

Output:

Enter your name: Tejaswini

Tejaswini

Note: By default the input is considered as a string if you pass anything.

Ex:

#perform arithematic addition by taking input values from the user or at runtime.

```
a=input("Enter a value:")
b=input("Enter b value.")
print("sum =",a+b)
```

Output:

Enter a value:10

Enter b value:20

Sum=1020

Note :

Here the a and b values arer considered as the strings by the input function and by using the concatenation method we get the output as 1020.

#To avoid this explicit type cast is required to drawback of input function:

```
a=int(input("Enter your  a value:"))
b=int(input("Enter your b value:"))
print("sum=",a+b)
```

Output:

Enter a value:10

Enter b value:20

Sum=30

Output Functions in Python

In Python, we use the print() function as the primary way to output data. Below are different ways to use the print() function effectively.

PYTHON PROGRAMMING LANGUAGE

Python is a high-level, interpreted, and object-oriented programming language known for its simplicity and readability. It was created by Guido van Rossum and first released in 1991. Python's design philosophy emphasizes code readability, using significant indentation to define code blocks, which makes it easier to understand and write.

Why python for beginners?

- Easy to learn(shorter code than java , c…).
- Open source(no need for license for using python, simply download it and use it)
- oop(object-oriented programming-which supports,inheritance,abstraction,polymorphism,encapsulation,exception handling , reusability of code is more, code is highly secured, modularity concept is proposed)
- procedural functional.
- Portable(write once run anywhere),GUI programming.
- Employed in broad range of applications rising demand for python programmers.

Python Interfaces and Installations:

IDLE , COLAB,JYUPTER

Features of python:

- High-level interpreted interactive language.
- Extensive standard libraries.
- Cross platform compatibility.
- Portable and extensible.
- Supports databases and gui programming
- Scalable and dynamic semantics
- Automatic garbage collection

Tokens of python:

Types of comments in python:

PYTHON PROGRAMMING LANGUAGE

There are two types of comments in python:

- Single-line comment:Represented with hashtag.

  Ex:#single-line comment

- Multi-line comment: represented using front slash and an astrick.

  Ex:/*this

  Is multi-line comment*/

The comments improve readability of the program. They use to document software requirements and associated information properly.

Types of Data in python:

13. Numeric data(integer,real ,float data)
14. String data(alphanumeric data)
15. Boolean data
16. None data(missing data)

Numeric data: The numeric data type in Python represents the data that has a numeric value. A numeric value can be an integer, a floating number, or even a complex number.

- Integers – This value is represented by int class. It contains positive or negative whole numbers (without fractions or decimals). In Python, there is no limit to how long an integer value can be.
- Float – This value is represented by the float class. It is a real number with a floating-point representation. It is specified by a decimal point. Optionally, the character e or E followed by a positive or negative integer may be appended to specify scientific notation.
- Complex Numbers – A complex number is represented by a complex class. It is specified as *(real part) + (imaginary part)j* . For example – 2+3j

Ex:

```
a = 5
print(type(a))
b = 5.0
print(type(b))
c = 2 + 4j
print(type(c))
```

output:

<class 'int'>

<class 'float'>

<class 'complex'>

2. Sequence Data Types in Python

The sequence Data Type in Python is the ordered collection of similar or different Python data types. Sequences allow storing of multiple values in an organized and efficient fashion.

- **String Data Type**

Python Strings are arrays of bytes representing Unicode characters. In Python, there is no character data type Python, a character is a string of length one. It is represented by str class.

Strings in Python can be created using single quotes, double quotes or even triple quotes. We can access individual characters of a String using index.

Ex:

```
s = 'Welcome to the Amar World'
print(s)
# check data type
print(type(s))
# access string with index
print(s[1])
print(s[2])
print(s[-1])
```

Output:

Welcome to the Amar World

<class 'str'>

e

l

d

- **List Data Type**

Lists are just like arrays, declared in other languages which is an ordered collection of data. It is very flexible as the items in a list do not need to be of the same type.

Creating a List in Python

Lists in Python can be created by just placing the sequence inside the square brackets[].

Ex: # Empty list

```
a = []

# list with int values

a = [1, 2, 3]

print(a)

# list with mixed int and string

b = ["Amar", "For", "Amar", 4, 5]

print(b)
```

Output:

[1, 2, 3]

[Amar, 'For', Amar, 4, 5]

Access List Items

In order to access the list items refer to the index number. In Python, negative sequence indexes represent positions from the end of the array. Instead of having to compute the offset as in List[len(List)-3], it is enough to just write List[-3]. Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second-last item, etc.

```
a = ["Ayan", "For", "Ayan"]

print("Accessing element from the list")

print(a[0])

print(a[2])

print("Accessing element using negative indexing")

print(a[-1])

print(a[-3])
```

Output

Accessing element from the list

Ayan

Ayan

Accessing element using negative indexing

Ayan

Ayan

- **Tuple Data Type**

Just like a list, a tuple is also an ordered collection of Python objects. The only difference between a tuple and a list is that tuples are immutable. Tuples cannot be modified after it is created.

Creating a Tuple in Python

In Python Data Types, tuples are created by placing a sequence of values separated by a 'comma' with or without the use of parentheses for grouping the data sequence. Tuples can contain any number of elements and of any datatype (like strings, integers, lists, etc.

```
# initiate empty tuple

tup1 = ()

tup2 = (Amit, 'For')

print("\nTuple with the use of String: ", t2)
```

Output

Tuple with the use of String:
(Amit, 'For')

Note *– The creation of a Python tuple without the use of parentheses is known as Tuple Packing.*

Access Tuple Items

In order to access the tuple items refer to the index number. Use the index operator [ ] to access an item in a tuple.

```
tup1 = tuple([1, 2, 3, 4, 5])

# access tuple items

print(tup1[0])

print(tup1[-1])

print(tup1[-3])
```

Output

1

5

3

- **Boolean Data Type**

Python Data type with one of the two built-in values, True or False. Boolean objects that are equal to True are truthy (true), and those equal to False are falsy (false). However non-Boolean objects can be evaluated in a Boolean context as well and determined to be true or false. It is denoted by the class bool.

Example: The first two lines will print the type of the boolean values True and False, which is **<class 'bool'>.** The third line will cause an error, because true is not a valid keyword in Python. Python is case-sensitive, which means it distinguishes between uppercase and lowercase letters.

```
print(type(True))
print(type(False))
```

Output:

```
<class 'bool'>
<class 'bool'>
```

- **None data(missing data)**

In Python, missing data is often represented by None in regular Python objects and NaN (Not a Number) in numerical computations using libraries like NumPy and Pandas.

Understanding None in Python

- None is a special keyword representing the absence of a value.
- It is **not** the same as 0, False, or an empty string ("").
- None is an instance of the NoneType class.

Example:

```
x = None
print(x)  # Output: None
print(type(x))  # Output: <class 'NoneType'>
```

Representation of raw data: binary, octal, hexadecimal, decimal.

#Declaration of binary data:

```
b=0b1010
c=0o124
h=0x12D
a=20
print(b)
print(c)
print(h)
print(a)
```

Output:10 84 301 20

Input& Output functions:

Input function:

A=20#this is a static declaration of a variable value.

Dynamically input can be taken from the user using input function.

Ex:

```
name = input("Enter your name")
print(name)
```

Output:

Enter your name: Tejaswini

Tejaswini

Note: By default the input is considered as a string if you pass anything.

Ex:

#perform arithematic addition by taking input values from the user or at runtime.

```
a=input("Enter a value:")
b=input("Enter b value.")
print("sum =",a+b)
```

Output:

Enter a value:10

Enter b value:20

Sum=1020

Note :

Here the a and b values arer considered as the strings by the input function and by using the concatenation method we get the output as 1020.

#To avoid this explicit type cast is required to drawback of input function:

```
a=int(input("Enter your  a value:"))
b=int(input("Enter your b value:"))
print("sum=",a+b)
```

Output:

Enter a value:10

Enter b value:20

Sum=30

Output Functions in Python

In Python, we use the print() function as the primary way to output data. Below are different ways to use the print() function effectively.

Ex:

```
name = "Alice"
age = 25
print("Name:", name, "Age:", age)
```

Output: Name: Alice Age: 25

Ex:

```
name = "Alice"
age = 25
print("Name:", name, "Age:", age)
```

Output: Name: Alice Age: 25

Ex:

```
name = "Alice"
age = 25
print("Name:", name, "Age:", age)
```

Output: Name: Alice Age: 25

Ex:

```
name = "Alice"
age = 25
print("Name:", name, "Age:", age)
```

Output: Name: Alice Age: 25

Interface or software to execute Python programming languages to implement Data science.

Here's how to download and install each of these Python IDEs:

1. **IDLE**
   - Included by default with Python.
   - Download Python from python.org and install it.
   - Open IDLE from the start menu or terminal.

-

2. **Google Colab**
   - No installation needed.
   - Visit colab.research.google.com and sign in with a Google account.
   - Start a new notebook to write and run Python code.
3. **Spyder**
   - Download Anaconda from anaconda.com.
   - Install Anaconda, which includes Spyder.
   - Open Spyder from the Anaconda Navigator.
4. **PyCharm**
   - Download from jetbrains.com/pycharm.
   - Choose Community (free) or Professional (paid) version.
   - Install and open PyCharm to start coding.
5. **Canopy**
   - Download from enthought.com/products/canopy.
   - Install and open Canopy for scientific computing.
   - Requires registration for full features.
6. **Thonny**
   - Download from thonny.org.
   - Install and open Thonny.
   - Best for beginners and simple projects.
7. **Atom**
   - Download from atom.io.
   - Install and open Atom.
   - Install Python support via the "script" or "ide-python" plugin.

Operators:

An Operator is a symbol notation which perform a specific operation based on given operands.

- **Arithmetic Operator(+,-,*,/,%,**,//):**

Ex: # Addition

```
a = 10 + 5
print("Addition:", a)
# Subtraction
b = 10 - 5
print("Subtraction:", b)
# Multiplication
c = 10 * 5
print("Multiplication:", c)
# Division
d = 10 / 5
print("Division:", d)
# Floor Division (removes decimal)
e = 10 // 3
print("Floor division:", e)
 # Modulus (remainder)
f = 10 % 3
print("Remainder:" f)


# Exponentiation (power)
g = 2 ** 3
print("Exponentiation:", g)
```

```
Output: Addition: 15
Subtraction: 5
Multiplication: 50
Division: 2.0
Floor division: 3
Remainder: 1
Exponentiation: 8
```

- **Relational Operator(< , <= ,> ,>=, == , !=):**

  Ex:

```
a = 10
b = 5
print("Equal to:", a == b)
print("Not equal to:", a != b)
print("Greater than:", a > b)
print("Less than:", a < b)
print("Greater than or equal to:", a >= b)
print("Less than or equal to:", a <= b)
```

  Output:

```
Equal to: False
Not equal to: True
Greater than: True
Less than: False
Greater than or equal to: True
Less than or equal to: False
```

- **Logical Operator(and, or, not):**

  Ex:

```
x = True
y = False
print("AND:", x and y)
print("OR:", x or y)
print("NOT:", not x)
```

  Ouptut:

```
AND: False
OR: True
NOT: False
```

- **Assignment Operator(=, +=, -=, *=, /=)**

  Ex:

```
a = 10
a += 5
```

```
print("Addition Assignment:", a)
a -= 3
print("Subtraction Assignment:", a)
a *= 2
print("Multiplication Assignment:", a)
a /= 4
print("Division Assignment:", a)
a %= 3
print("Modulus Assignment:", a)
```

Output:

Addition Assignment: 15

Subtraction Assignment: 12

Multiplication Assignment: 24

Division Assignment: 6.0

Modulus Assignment: 0.0

- **Membership Operator(in, not in):**

  Ex:

```
lst = [1, 2, 3, 4, 5]
print("Is 3 in the list?", 3 in lst)
print("Is 6 not in the list?", 6 not in lst)
```

Output:

Is 3 in the list? True

Is 6 not in the list? True

- **Identity Operator(is, is not):**

  Ex:

```
a = 10
b = 10
c = [1, 2, 3]
d = [1, 2, 3]
print("a is b:", a is b)
print("c is d:", c is d)
print("c is not d:", c is not d)
```

Ouput:

a is b: True

c is d: False

c is not d: True

- **Bitwise Operator(&, \, ^,<<,>>,~):**

  Ex:

```
a = 5  # 0101 in binary
b = 3  # 0011 in binary
```

```
print("Bitwise AND:", a & b)
print("Bitwise OR:", a | b)
print("Bitwise XOR:", a ^ b)
print("Bitwise NOT:", ~a)
print("Left Shift:", a << 1)
print("Right Shift:", a >> 1)
```

Ouput: Bitwise AND: 1

Bitwise OR: 7

Bitwise XOR: 6

Bitwise NOT: -6

Left Shift: 10

Right Shift: 2