Python Programming

Python Introduction

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

It is used for:

- web development (server-side),
- software development,
- mathematics,
- system scripting.

Creating variables and assigning values

To create a variable in Python, all you need to do is specify the variable name, and then assign a value to it

<variable name> = <value>

- Python uses = to assign values to variables. There's no need to declare a variable in advance (or to assign a data
- type to it), assigning a value to a variable itself declares and initializes the variable with that value. There's no way to
- declare a variable without assigning it an initial value.

Example:

❖Integer

```
a = 2
print(a)
# Output: 2
```

❖String

```
a = “Apple“
print(a)
#Output: Apple
```

❖Float

```
a =3.2
print(a)
#Output: 3.2
```

Rules for variable naming:

1. Variables names must start with a letter or an underscore.

```
x = True # valid
 _y = True # valid
 9x = False # starts with numeral
=> SyntaxError: invalid syntax
 $y = False # starts with symbol
⇒SyntaxError: invalid syntax
```

2. The remainder of your variable name may consist of letters, numbers and underscores.

```
has_0_in_it = "Still Valid“
```

3. Names are case sensitive.

```
x = 9
y = X*5
```

=>NameError: name 'X' is not defined

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume). Rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)
- A variable name cannot be any of the Python keywords.

Python Variables - Assign Multiple Values

Python allows you to assign values to multiple variables in one line:

Example:

```
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)

# Output:

Orange

Banana

Cherry
```

One Value to Multiple Variables:

And you can assign the same value to multiple variables in one line:

Example:

```
x = y = z = "Orange"
print(x)
print(y)
print(z)
#Output:
 Orange
```

Python Data Types

- In programming, data type is an important concept.
- Variables can store data of different types, and different types can do different things.
- Python has the following data types built-in by default, in these categories:

Text Type : str
Numeric Types : int, float, complex
Sequence Types : list, tuple, range
Mapping Type : dict
Set Types : set, frozenset
Boolean Type : bool
Binary Types : bytes, bytearray, memoryview
None Type : NoneType

Sequence Types:

➢List

- Lists are used to store multiple items in a single variable.
- Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.
- Lists are created using square brackets

Example:

```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

List Items

- List items are ordered, changeable, and allow duplicate values.
- List items are indexed, the first item has index [0], the second item has index [1] etc.

Ordered

- When we say that lists are ordered, it means that the items have a defined order, and that order will not change.
- If you add new items to a list, the new items will be placed at the end of the list.

Changeable

- The list is changeable, meaning that we can change, add, and remove items in a list after it has been created.

Allow Duplicates

Since lists are indexed, lists can have items with the same value:

Example:

```
thislist = ["apple", "banana", "cherry", "apple", "cherry"]
print(thislist)
```

Output:

['apple', 'banana', 'cherry', 'apple', 'cherry’]

List Length

To determine how many items a list has, use the len() function:

Example:

```
thislist = ["apple", "banana", "cherry"]
print(len(thislist))
```

Output: 3

type()

Example:

```
mylist = ["apple", "banana", "cherry"]
print(type(mylist))
```

Output:<class 'list’>

Access Items

List items are indexed and you can access them by referring to the index number:

Example:

Print the second item of the list:

```
thislist = ["apple", "banana", "cherry"]
print(thislist[1])
```

Output:

banana

Change Item Value

To change the value of a specific item, refer to the index number:

```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)
```

Output:['apple', 'blackcurrant', 'cherry’]

Add List Items

To add an item to the end of the list, use the append() method:

```
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)
```

Output:

['apple', 'banana', 'cherry', 'orange']

Insert Items

To insert a list item at a specified index, use the insert() method:

Example:

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(1, "orange")
print(thislist)
```

Output : ['apple', 'orange', 'banana', 'cherry']

Extend List

To append elements from another list to the current list ,use extend() method.

Example:

```
thislist = ["apple", "banana", "cherry"]
tropical = ["mango", "pineapple", "papaya"]
thislist.extend(tropical)
print(thislist)
```

Output:

['apple', 'banana', 'cherry', 'mango', 'pineapple', 'papaya']

Remove List Items

The remove() method removes the specified item.

Example:

```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
```

Output:

['apple', 'cherry']

Remove Specified Index

The pop() method removes the specified index

Example:

```
thislist = ["apple", "banana", "cherry"]
thislist.pop(1)
print(thislist)
```

Output:

['apple', 'cherry']

➢ Tuple

- Tuples are used to store multiple items in a single variable.
- A tuple is a collection which is ordered and unchangeable.
- Tuples are written with round brackets.

Example:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

Output:('apple', 'banana', 'cherry')

Tuple Items:

- Tuple items are ordered, unchangeable, and allow duplicate values.
- Tuple items are indexed, the first item has index [0], the second item has index [1] etc.

Tuple Items

- Tuple items are ordered, unchangeable, and allow duplicate values.
- Tuple items are indexed, the first item has index [0], the second item has index [1] etc.

Ordered

When we say that tuples are ordered, it means that the items have a defined order, and that order will not change.

Unchangeable

Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.

Allow Duplicates

Since tuples are indexed, they can have items with the same value

➢ Range

A range is a sequence of numbers, commonly used for looping a specific number of times in for loops.

Example 1:

```
x = range(0,10)
print( list(x))
#output:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Example 2:

```
for i in range(0,5):
    print(i)
#output
0
1
2
3
4
```

Mapping Type :Dictionary

Dictionaries are used to store data values in key:value pairs.

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

Example:

```
cars= {"brand":"Ford","model":"Mustang","year":1964}
print(cars)
#Output: {'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

Dictionary Items:

- Dictionary items are ordered, changeable, and do not allow duplicates.
- Dictionary items are presented in key:value pairs, and can be referred to by using the key name.

Example:

```
thisdict = {  "brand": "Ford",  "model": "Mustang",  "year": 1964}
print(thisdict["brand"])
#output: Ford
```

Ordered

As of Python version 3.7, dictionaries are ordered. In Python 3.6 and earlier, dictionaries are unordered.

- When we say that dictionaries are ordered, it means that the items have a defined order, and that order will not change.
- Unordered means that the items do not have a defined order, you cannot refer to an item by using an index.

Changeable

Dictionaries are changeable, meaning that we can change, add or remove items after the dictionary has been created.

Duplicates Not Allowed

Dictionaries cannot have two items with the same key:

Example:

Duplicate values will overwrite existing values:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964,
  "year": 2020
}
print(thisdict)
#output:{'brand': 'Ford', 'model': 'Mustang', 'year': 2020}
```

- Write a Python program that adds two numbers entered by the user and displays the result along with the data type of the result.

```
num1 = input("Enter first number: ")
num2 = input("Enter second number: ")

# Convert to float to handle both int and decimal input
result = float(num1) + float(num2)

print("Sum:", result)
print("Data type of result:", type(result))
```