Python Roadmap

Python Programming

1. Basics to Advanced
2. Conditional and Control statements, Loops
3. Functions & Oops

Essential Python Libraries

1. Numpy
2. Pandas
3. Matplotlib
4. Seaborn

- File handling & DateTime
- Regular Expressions
- Web scraping

Why we use jupyter notebook???

Jupyter notebook, formerly known as the IPython(Interactive Python) notebook, is a flexible tool that helps you create readable analyses, as you can keep code, images, comments, formulae and plots together

The name Jupyter is an indirect acronyum of the three core languages it was designed for: **JU**lia, **PYT**hon, and **R** and is inspired by the planet Jupiter.

Keyboard shortcuts

1. **Command Mode (press Esc to unable)**

`Enter` - enter edit mode
`shift + enter` - run cell , select below
`ctrl + enter` - run cell
`Alt + enter` - run cell, insert below
`Y` - to code
`M` - to markdown
`R` - to raw
`1` - to heading 1
`23456` - to heading 23456
`A` / `B` - insert above and below cell
`X` - cut selected cell
`C` - copy selected cell
`shift + v` - paste cell above
`v` - paste cell below
`z` - undo last cell deletion
`D` - delet selected cell
`shift + M` - merge cell below
`ctrl + S` - save and checkpoint
`L` - toggle line number
`o` - toggle output
`shift + o` - toggle output scrolling
`Esc` - close pager
`o` - restart kernel
`space` - scroll down
`shift + space` - scroll up
`shift` - ignore

2. **Edit Mode (press Enter to enable)**

`tab` - code completion or indent
`shift` + `tab` - tooltip
`ctrl + ]` - indent
`ctrl + [` - denent
`ctrl + A` - select all
`ctrl + shift + z` - redo
`ctrl + y` - redo
`ctrl + Home` - go to cell start
`ctrl + up` - go to cell start
`ctrl + End` - go to cell end
`ctrl + Down` - go to cell end
`ctrl + left` - go one word left
`ctrl + right` - go one word right
`ctrl + backspace` - delet word before
`ctrl + delet` - delet word after
`ctrl + M` - command mode
`ctrl` + `shift` + `minus` - split cell
`ctrl + /` - toggle comment on current or selected lines

Python variables

A variable is a reserved memory area (memory address) to store value. For example, we want to store an employee's salary. In such a case, we can create a variable and store salary using it. Using that variable name, you can read or modify the salary amount.

In other words, a variable is a value that varies according to the condition or input pass to the program. Everything in Python is treated as an object so every variable is nothing but an object in Python.

A variable can be either mutable or immutable. If the variable's value can change, the object is called mutable, while if the value cannot change, the object is called immutable.

Creating a variable

Python programming language is dynamically typed, so there is no need to declare a variable before using it or declare the data type of variable like in other programming languages. The declaration happens automatically when we assign a value to the variable.

Creating a variable and assigning a value

We can assign a value to the variable at that time variable is created. We can use the assignment operator = to assign a value to a variable.

The operand, which is on the left side of the assignment operator, is a variable name. And the operand, which is the right side of the assignment operator, is the variable's value.

variable_name = variable_value

In [1]:
```
name = "Deepali"  # string assignment
age = 25  # integer assignment
salary = 25800.60  # float assignment

print(name)  # Deepali
print(age)  # 25
print(salary)  # 25800.6
```

```
Deepali
25
25800.6
```

In the above example, "Deepali", 25, 25800.60 are values that are assigned to name, age, and salary respectively.

Changing the value of a variable

Many programming languages are statically typed languages where the variable is initially declared with a specific type, and during its lifetime, it must always have that type.

But in Python, variables are dynamically typed and not subject to the data type restriction. A variable may be assigned to a value of one type, and then later, we can also re-assigned a value of a different type.

In [2]:
```
var = 10
print(var)  # 10
# print its type
print(type(var))  # <class 'int'>

# assign different integer value to var
var = 55
print(var)  # 55

# change var to string
var = "Now I'm a string"
print(var)  # Now I'm a string
# print its type
print(type(var))  # <class 'str'>

# change var to float
var = 35.69
print(var)  # 35.69
# print its type
print(type(var))  # <class 'float'>
```

```
10
<class 'int'>
55
Now I'm a string
<class 'str'>
35.69
<class 'float'>
```

Create Number, String, List variables

We can create different types of variables as per our requirements.

Number

A number is a data type to store numeric values. The object for the number will be created when we assign a value to the variable. In Python3, we

can use the following three data types to store numeric values.

1. Int
2. float
3. complex

Integer variable

The int is a data type that returns integer type values (signed integers); they are also called ints or integers. The integer value can be positive or negative without a decimal point.

In [3]:
```
# create integer variable
age = 24
print(age)  # 24
print(type(age))  # <class 'int'>
```

```
24
<class 'int'>
```

Note: We used the built-in Python method type() to check the variable type.

Float variable

Floats are the values with the decimal point dividing the integer and the fractional parts of the number. Use float data type to store decimal values.

In [4]:
```
# create float variable
salary = 10800.55
print(salary)  # 10800.55
print(type(salary))  # <class 'float'>
```

```
10800.55
<class 'float'>
```

In the above example, the variable salary assigned to value 10800.55, which is a float value.

Complex type

The complex is the numbers that come with the real and imaginary part. A complex number is in the form of a+bj, where a and b contain integers or floating-point values.

In [5]:
```
a = 9 + 5j
print(a)  # (9+5j)
print(type(a))  # <class 'complex'>
```

```
(9+5j)
<class 'complex'>
```

String variable

In Python, a string is a set of characters represented in quotation marks. Python allows us to define a string in either pair of single or double quotation marks. For example, to store a person's name we can use a string type.

To retrieve a piece of string from a given string, we can use to slice operator [] or [:]. Slicing provides us the subset of a string with an index starting from index 0 to index end-1.

To concatenate the string, we can use the addition(+) operator.

In [6]:
```
# create a variable of type string
str = 'DataScience'
# prints complete string
print(str)  # Data Science

# prints first character of the string
print(str[0])  # D

# prints characters starting from 2nd to 5th
print(str[2:5])  # taS

# length of string
print(len(str))  # 12

# concatenate string
print(str + "TEST")  # Data ScienceTEST
```

```
DataScience
D
taS
11
DataScienceTEST
```

Print

In [5]:
```
print("hi    how      are     you")
```

```
hi    how      are     you
```

In [6]:
```
print("Hi" + "Hello")
```

```
HiHello
```

In [7]:
```
print("Hi"+ " "+ "Hello")
```

```
Hi Hello
```

In [8]:
```
print("Hyderabad" * 3)  # its prints string multiple times
```

```
HyderabadHyderabadHyderabad
```

In [9]:
```
print("I multipy", 2 * 3 , "Is the result") # print multiply with  numbers
```

```
I multipy 6 Is the result
```

multi lines strings

In [10]:
```
print('''To get the list of all keywords of Python programmatically, you can use kwlist of keyword library.
Following is the quick code snippet to get the list of all keywords.
kwlist returns sequence containing all the keywords defined for the interpreter.''')
```

```
To get the list of all keywords of Python programmatically, you can use kwlist of keyword library.
Following is the quick code snippet to get the list of all keywords.
kwlist returns sequence containing all the keywords defined for the interpreter.
```

print with space , \n , \t

In [11]:
```
print(" Hi How are you?")
```

```
 Hi How are you?
```

In [12]:
```
print( " hi \nHow are you")
```

```
 hi
How are you
```

In [14]:
```
print("hi\thiiiiiiiiiii")
```

```
hi      hiiiiiiiiiii
```

In [15]:
```
1 print('I don't know')
```

```
  File "<ipython-input-15-ffef336255b4>", line 1
    print('I don't know')
                  ^
SyntaxError: invalid syntax
```

In [16]:
```
1 print('I don\'t know')
```

```
I don't know
```

In [18]:
```
1 print("\"Hi\", He Said")
```

```
"Hi", He Said
```

In [19]:
```
1 pi = 3.141592653
```

In [20]:
```
1 print("I am pi %d"%(pi)) # %d is place holder for Integers
```

```
I am pi 3
```

In [21]:
```
1 print("I am pi %f"%(pi)) # %f is place holder for Integers
```

```
I am pi 3.141593
```

In [22]:
```
1 print("I am pi %.3f"%(pi)) # %f is place holder for Integers
```

```
I am pi 3.142
```

In [23]:
```
x = 25
y = 2
tempScale = 'Celsius'
```

In [24]:
```
print("The temperature is %.2f %s"%(x/y, tempScale))
```

```
The temperature is 12.50 Celsius
```

In [25]:
```
a = 5
b = 6
print("I am FIVE {}, I am SIX {}".format(a,b))
```

```
I am FIVE 5, I am SIX 6
```

In [26]:
```
print("I am pi {}".format(pi))

```

```
I am pi 3.141592653
```

In [27]:
```
print("I am pi {:.3f}".format(pi))
```

```
I am pi 3.142
```

If we want to represent a group of elements (or value) as a single entity, we should go for the list variable type. For example, we can use them to store student names. In the list, the insertion order of elements is preserved. That means, in which order elements are inserted in the list, the order will be intact.

The list can be accessed in two ways, either positive or negative index. The list has the following characteristics:

1. In the list insertion order of elements is preserved.
2. Heterogeneous (all types of data types int, float, string) elements are allowed.
3. Duplicates elements are permitted.
4. The list is mutable(can change).
5. Growable in nature means based on our requirement, we can increase or decrease the list's size.
6. List elements should be enclosed within square brackets [].

Variable's case-sensitive

Python is a case-sensitive language. If we define a variable with names a = 100 and A =200 then, Python differentiates between a and A. These variables are treated as two different variables (or objects).

In [8]:
```
a = 100
A = 200

# value of a
print(a)  # 100
# Value of A
print(A)  # 200

a = a + A
print(a)  # 300
```

```
100
200
300
```

Constant

Constant is a variable or value that does not change, which means it remains the same and cannot be modified. But in the case of Python, the constant concept is not applicable. By convention, we can use only uppercase characters to define the constant variable if we don't want to change it.

In [9]:
```
MIN_VALUE = 500
# It is just convention, but we can change the value of MAX_VALUE variable.
```

Rules and naming convention for variables and constants

A name in a Python program is called an identifier. An identifier can be a variable name, class name, function name, and module name.

There are some rules to define variables in Python.

In Python, there are some conventions and rules to define variables and constants that should follow.

- Rule 1: The name of the variable and constant should have a combination of letters, digits, and underscore symbols.

Alphabet/letters i.e., lowercase (a to z) or uppercase (A to Z) Digits(0 to 9) Underscore symbol (_)

For Example :

- total_addition
- TOTAL_ADDITION
- totalAddition
- Totaladdition

- Rule 2: The variable name and constant name should make sense.

Note: we should always create a meaningful variable name so it will be easy to understand. That is, it should be meaningful.

For Example: x = "Deepali"
student_name = "Deepali"

It above example variable x does not make more sense, but student_name is a meaningful variable.

- Rule 3: Don't' use special symbols in a variable name

For declaring variable and constant, we cannot use special symbols like $, #, @, %, !~, etc. If we try to declare names with a special symbol, Python generates an error

Example

- ca$h = 1000

- Rule 4: Variable and constant should not start with digit letters.

You will receive an error if you start a variable name with a digit. Let's verify this using a simple example.

- 1studnet = "Deepali"
- print(1studnet)
- Here Python will generate a syntax error at 1studnet. instead of this, you can declare a variable like studnet_1 = "Deepali"

- Rule 5: Identifiers are case sensitive.

In [10]:
```
total = 120
Total = 130
TOTAL = 150
print(total)
print(Total)
print(TOTAL)

```

```
120
130
150
```

Here, Python makes a difference between these variables that is uppercase and lowercase, so that it will create three different variables total, Total, TOTAL.

- Rule 6: To declare constant should use capital letters.

In [11]:
```
MIN_VALUE = 100
MAX_VALUE = 1000
```

Use an underscore symbol for separating the words in a variable name

If we want to declare variable and constant names having two words, use an underscore symbol for separating the words.

In [12]:
```
current_temperature = 24
```

Assigning multiple values to multiple variables

In [1]:
```
roll_no, marks, name = 10, 70, "Deepali"
print(roll_no, marks, name) # 10 20 Deepali
```

```
10 70 Deepali
```

In the above example, two integer values 10 and 70 are assigned to variables roll_no and marks, respectively, and string literal, “Deepali,” is assigned to the variable name.

Quick Summary to keep in mind:

Checklist while giving variable names

- Allowed values -> alphabets, numbers and underscore (_).
- A variable name should start with alphabet or underscore. It can not start with a number.
- Variable names are case-sensitive.

In [ ]:
```

```