Introduction to Python

Understanding the Basics

Variables in Python

- Suppose there is a girl in the class named Kavya. Our task is to store her name in Python. How can we do that?
- The **studentname** is known as variable or identifier.

```
[1]: studentname = 'Kavya'
```

What are variables?

- In Python, variables are used to store data values.

- A variable can hold different types of data, such as numbers, strings, lists, or more complex data structures.

- Unlike some other programming languages, Python does not require you to declare the type of variable before using it, as it is **dynamically typed**. This means you can assign a value to a variable without specifying its data type, and the type is determined automatically based on the assigned value.

Assigning a value in the variable

- To assign the value in the variable we can use the equal to sign (=)
- Few examples for your understanding

```
[1]: studentname = 'Kavya'

[3]: studentsubject = 'Mathematics'

[5]: studentmarks = 95

[7]: studentpercentage = 94.5

[9]: studentpass = True
```

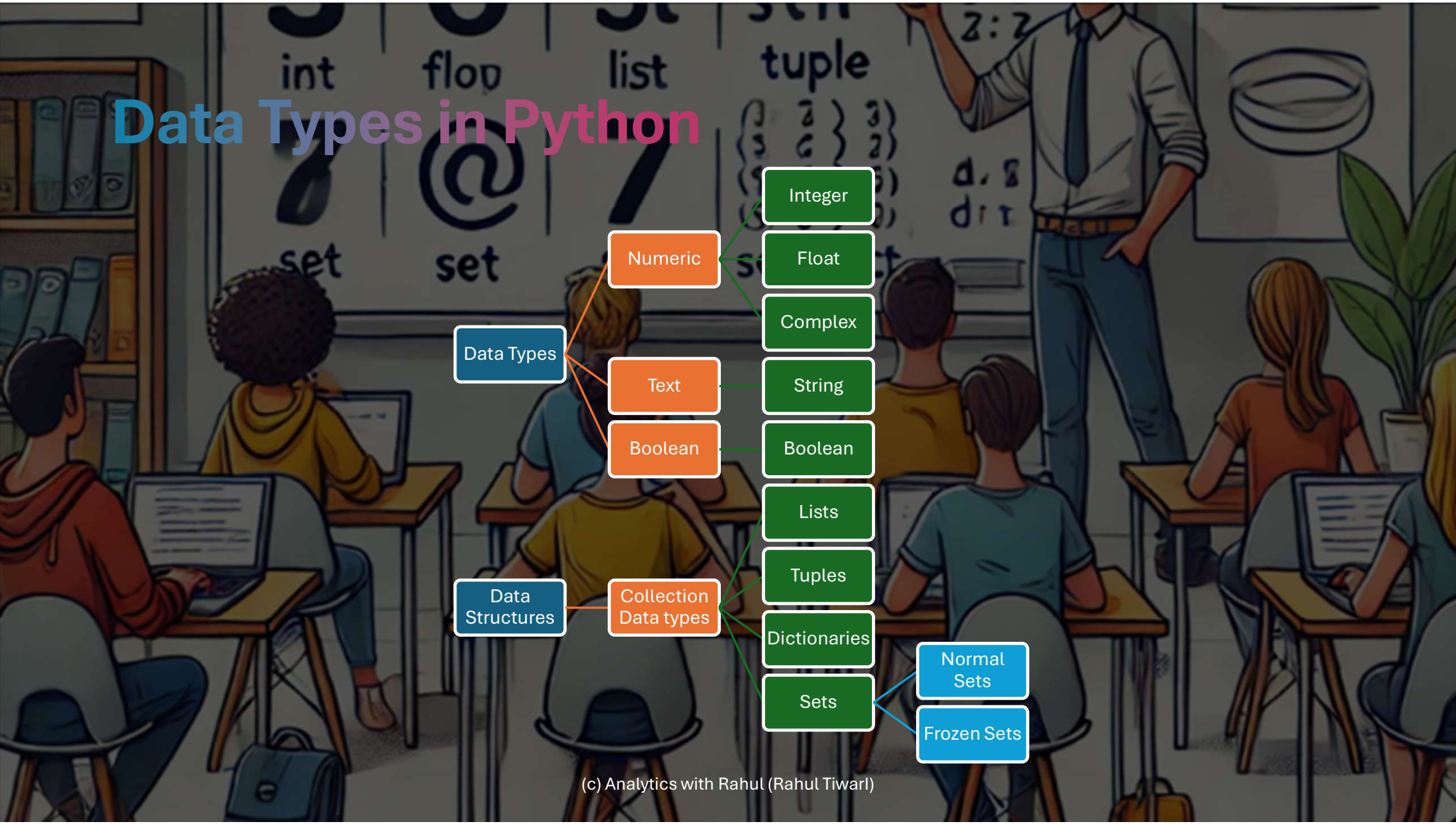
Where are the values getting stored?

- In Python, the **id()** function is used to get the unique identifier (memory address) of an object. The unique identifier is an integer that remains constant for the object during its lifetime. This identifier can be used to compare the memory addresses of two objects to see if they are the same object in memory.

```
[1]: studentname = 'Kavya'

[11]: id(studentname)

[11]: 2514887304432

[3]: studentsubject = 'Mathematics'

[13]: id(studentsubject)

[13]: 2514887272880
```

Naming convention of a variable

- Variable names must start with a letter (a-z, A-Z) or an underscore (_).
- Subsequent characters can be letters, numbers (0-9), or underscores.
- Numbers cannot be at the start.
- Variable names are case-sensitive (e.g., age and Age are different).
- All the Python specific keywords we cannot use as variable names. To get the list of keywords you can use **keyword module**.
- For all the constant variables like PI π, may be in the business point of view INTERESTRATE etc., we keep that variables as capitals.


Data Types in Python
Data Types
Numeric
Integer
Float
Complex
Text
String
Boolean
Boolean
Data Structures
Collection Data types
Lists
Tuples
Dictionaries
Sets
Normal Sets
Frozen Sets
(c) Analytics with Rahul (Rahul Tiwari)

Input Function

In Python, the **input()** function is used to take input from the user. It reads a line from the standard input (usually the keyboard) and returns it as a **string.** The **input()** function pauses the program and waits for the user to enter some input, which can then be used in the program.

```
[*]: name = input('What is your name?')
What is your name?
```

Input Function

- When using the **input()** function in Python, the data entered by the user is stored as a string by default. If we need to store the data in a different datatype, we must apply typecasting either at the time of input or later to convert the data to the desired type.

```
[31]: marks = input('Enter the marks: ')
      marks

      Enter the marks:  10
[31]: '10'
```

```
[29]: marks = int(input('Enter the marks: '))
      marks

      Enter the marks:  10
[29]: 10
```

```
[33]: marks = input('Enter the marks: ')
      int(marks)

      Enter the marks:  10
[33]: 10
```

Output Function

- In Python, the **print()** function is used to output information to the console or standard output. It is a built-in function that can display text, numbers, variables, and other data types. The **print()** function is commonly used for debugging, displaying results, and providing user-friendly messages in programs.

```
[35]: print("Hello, World!")
```

```
Hello, World!
```

```
[37]: name = "Alice"
      age = 30
      print("Name:", name, "Age:", age)
```

```
Name: Alice Age: 30
```

```
[39]: print("Python", "Tableau", "SQL", sep=", ")
```

```
Python, Tableau, SQL
```

```
[41]: print("Hello", end="!")
      print("World")
```

```
Hello!World
```

Math Module

- The math module in Python is a standard library module that provides a wide range of mathematical functions and constants. It includes functions for basic arithmetic, trigonometry, logarithms, exponentiation, and more advanced mathematical operations.

```
[47]: import math
      print(math.sqrt(16))
      print(math.pi)
      print(math.log(100, 10))
      print(math.factorial(5))
      print(math.pow(10,3))
```

```
4.0
3.141592653589793
2.0
120
1000.0
```

Operators in Python

Arithmetic Operators

(+) Sum: Adds two or more numbers

(-) Subtraction: Subtracts two or more numbers

(*) Multiplication: Multiplies two or more numbers

(/) Division: Divide two or more numbers

(//) Floor Division: It helps in getting the quotient

(%) Modulo: It helps in getting the remainder

(**) Power: It helps to get the power

```
[67]: num1 = 10
      num2 = 3
      print('Sum: ', num1+num2)
      print('Subtraction: ', num1-num2)
      print('Multiplication: ', num1*num2)
      print('Division: ', num1/num2)
      print('Floor Division: ', num1//num2)
      print('Modulo: ', num1%num2)
      print('Power: ', num1**num2)
```

```
Sum:  13
Subtraction:  7
Multiplication:  30
Division:  3.3333333333333335
Floor Division:  3
Modulo:  1
Power:  1000
```

Assignment Operators

(+=) Sum and assign

(-=) Subtraction and assign

(*=) Multiplication and assign

(/=) Division and assign

(//=) Floor Division and assign

(%=) Modulo and assign

(**=) Power and assign

```
[75]: num1 = 10
      num2 = 3
      num1+=num2
      print(num1)
```

```
13
```

Comparison Operators

- Comparison operators are used to compare two values and return a single Boolean value, which can be either True or False.

```
[83]: num1 = 10
      num2 = 3
      print('num1 is equal to num2: ',a==b)
      print('num1 is not equal to num2: ',a!=b)
      print('num1 is greater than num2: ',a>b)
      print('num1 is less than num2: ',a<b)
      print('num1 is greater than equal to num2: ',a>=b)
      print('num1 is less than equal to num2: ',a<=b)
```

```
num1 is equal to num2:  False
num1 is not equal to num2:  True
num1 is greater than num2:  True
num1 is less than num2:  False
num1 is greater than equal to num2:  True
num1 is less than equal to num2:  False
```

Logical Operators

- Logical operators, including **and, or,** and **not,** are used in conditional statements to determine whether a condition is True or False.

```
[91]: num1 = 10
      num2 = 3
      print((num1>2) and (num2<=1))
```

```
False
```

```
[93]: print(True and True)
      print(True and False)
      print(True or True)
      print(True or False)
      print(not True)
```

```
True
False
True
True
False
```

Special Operators

- Special Operators are of two types
  - **Membership Operators**
  - In Python, membership operators are used to test whether a value or variable is found in a sequence (such as a string, list, tuple, set, or dictionary). There are two membership operators:

  - **in:** Returns True if the specified value is present in the sequence; otherwise, it returns False.
  - **not in:** Returns True if the specified value is not present in the sequence; otherwise, it returns False.

```
[95]: names = ['James','Andrew','Sarah','Cynthia','Robert','John','Trent']
      'Rahul' in names

[95]: False
```

Special Operators

- **Identity Operators**
- In Python, identity operators are used to compare the **memory locations** of two objects. They determine whether two variables refer to the same object in memory, not just if they have the same value. There are two identity operators:

- **is:** Returns True if both variables point to the same object (i.e., they are identical).
- **is not:** Returns True if the variables point to different objects (i.e., they are not identical).

```
[97]: num1 = 10
      num2 = 10
      print(id(num1))
      print(id(num2))
      num1 is num2

      140720054287064
      140720054287064
[97]: True
```

Strings in Python

- In Python, strings are sequences of characters used to store text. They are immutable, meaning that once a string is created, its contents cannot be altered. However, Python offers numerous methods for manipulating and interacting with strings, making them one of the most commonly used data types.
- You can convert other types of data, such as integers, floats, and other objects, into strings by using the str() function. This conversion is often useful when you need to combine text with numerical data.

```
[99]: name = 'Rahul'

[101]: name

[101]: 'Rahul'

[105]: studentname = "Kavya"

[107]: studentname

[107]: 'Kavya'

[109]: studentdetails = '''The student is good in Mathematics
       and overall good performer'''

[111]: studentdetails

[111]: 'The student is good in Mathematics \nand overall good performer'

[113]: studentaddress = """Gurgaon, Sector 45,
       Haryana, 122001"""

[115]: studentaddress

[115]: 'Gurgaon, Sector 45, \nHaryana, 122001'

[117]: age = 38
       str(age)

[117]: '38'
```

String Formatting

- Python offers several functions for formatting strings, enabling the inclusion of variables within a string statement. These methods include the percent **(%) operator, the .format() method, and f-strings.**

- Introduced in Python 3.6, **f-strings** provide a concise, readable, and efficient method for formatting strings. By prefixing the string with an f or F, variables or expressions can be directly embedded within curly braces { }, allowing for easy integration of data into the string.

```
[119]: name = 'Kavya'
       marks = 95
       print(f'My name is {name} and I got {marks} marks in Mathematics')

       My name is Kavya and I got 95 marks in Mathematics
```

String Formatting

- The **.format()** function is a built-in method for formatting strings in Python. Both **.format()** and **f-strings** (formatted string literals) are used for string formatting, but they differ in syntax and characteristics. Here's a comparison of their similarities and differences:
- Below are the similarities

Feature	Description
Value Injection	Both .format() and f-strings are used to insert values into string placeholders, enabling the creation of dynamic strings.
Reordering and Insertion	They allow for the reordering, formatting, and insertion of values into strings without manual concatenation, enhancing code readability and maintainability.
Formatting Capabilities	Both methods provide features like setting decimal places, padding, alignment, and more, using similar syntax within the placeholders.

String Formatting

- Below are the Differences

Aspect	.format()	f-strings
Syntax	Uses placeholders inside the string marked by curly braces {}, replaced by values or variables passed to the .format() method.	Uses prefixed strings with f or F and directly embeds expressions inside curly braces {} within the string.
Performance	Generally slower because it involves method calls and is evaluated at runtime.	Faster, as they are evaluated at runtime with less overhead.
Readability	May require more verbose code if many variables are involved, as variables must be passed explicitly.	More readable, with variables and expressions directly embedded within the string, making the final output easier to understand.
Scope Usage	Requires passing variables explicitly to the method.	Can directly access variables and functions from the surrounding scope within the curly braces.

String Formatting

- .format() examples

[127]:
```
name = 'Rahul'
company = 'Deloitte'
salary = 10000
print('My name is {}, my salary is {} and my company is {}.'.format(name, salary, company))
```

```
My name is Rahul, my salary is 10000 and my company is Deloitte.
```

[129]:
```
name = 'Rahul'
company = 'Deloitte'
salary = 10000
print('My name is {0}, my salary is {1} and my company is {2}.'.format(name, salary, company))
```

```
My name is Rahul, my salary is 10000 and my company is Deloitte.
```

[131]:
```
name = 'Rahul'
company = 'Deloitte'
salary = 10000
print('My name is {n}, my salary is {s} and my company is {c}.'.format(n=name, s=salary, c=company))
```

```
My name is Rahul, my salary is 10000 and my company is Deloitte.
```

String Formatting

- The **% operator,** influenced by C's printf() format specifiers, provides an alternative method for formatting strings in Python. While it's less favored in contemporary Python code, it remains prevalent, particularly in older codebases. This method requires specifying the type for each variable being inserted into the string.

[145]:
```
name = 'Rahul'
percent = 10.25
salary = 10000
print('Hi I name is %s\nThe total hike which I got is %.2f\nMy current salary is %d' % (name, percent, salary))
```

```
Hi I name is Rahul
The total hike which I got is 10.25
My current salary is 10000
```

Escape Sequences and Characters

- When working with strings that include single and double quotes, or when special characters need to be inserted, Python utilizes escape sequences to manage these complexities.
- To create an escape sequence, use a backslash '' followed by the desired character. For instance, '\n' represents a new line, and '\t' represents a tab.
- Python strings can be enclosed in either single or double quotes. However, if a string contains both types of quotes, it may cause a Syntax Error. Consider the following example.

```
[153]: message = "Rahul asked me, "Hello what's going on?""
       print(message)

         Cell In[153], line 1
           message = "Rahul asked me, "Hello what's going on?""
                                                              ^
       SyntaxError: unterminated string literal (detected at line 1)

[151]: message = '''Rahul asked me, "Hello what's going on?"'''
       print(message)

       Rahul asked me, "Hello what's going on?"

[155]: message = "Rahul asked me, \"Hello what's going on?\""
       print(message)

       Rahul asked me, "Hello what's going on?"

[159]: message = 'Rahul asked me, "Hello what\'s going on?"'
       print(message)

       Rahul asked me, "Hello what's going on?"
```

Escape Sequences and Characters

- \n: This represents a new line. When included in a string, it moves the cursor to the start of the next line.

```
[167]: print('Hello\nWorld!')

Hello
World!
```

- \t: This represents a tab character, which inserts a horizontal space in the text.

```
[169]: print('Hello\tWorld!')

Hello	World!
```

- \\: This signifies a backslash character. It enables you to include an actual backslash in a string.

```
[161]: print('Backslash: \\')

Backslash: \
```