Python Revision Tour

Python character set

The character set refers to the valid characters that are used in Python.

- **Letters:** a-z, A-Z
- **Digits:** 0-9
- **Special Characters:** +, -, *, /, =, <, >, etc.
- **Whitespace:** Space, tab, and newline (\n)
- **Unicode Characters:** Python supports Unicode from various languages.

Token in Python

The smallest unit in a Python program is called a token. In python all the instructions and statements in a program are built with tokens. Different type of tokens in python are keywords, identifier, Literals/values, operators and punctuators –

- **Keywords:** Words with a special meaning in a programming language are known as keywords. There are 33 keywords in Python True, False, class, break, continue, and, as, try, while, for, or, not, if, elif, print, etc.
- **Identifiers:** Identifier is a user-defined name given to a variable, function, class, module, etc. Python has certain guidelines for naming identifiers, follow these guidelines:
  - Code in python is case – sensitive
  - Always identifier starts with capital letter(A – Z), small letter (a – z) or an underscore(_).
  - Digits are not allowed to define in first character, but you can use between.
  - No whitespace or special characters are allowed.
  - No keyword is allowed to define identifier.
- **Literals:** Literals are the raw data that is assigned to variables or constants during programming. There are five different types of literals string literals, numeric literals, Boolean literals and special literals none.
  - **String literals:** The string literals in Python are represented by text enclosed in single, double, or triple quotations. Examples include “Computer Science,” ‘Computer Science’, “’Computer Science”’ etc.
  - **Numeric Literals:** Literals that have been used to storing numbers is known is Numeric Literals. There are basically three numerical literals – Integer, Float and Complex.
  - **Boolean Literal:** Boolean literals have only two values True of False.
  - **Special literals none:** The special literal “None” in Python used to signify no values, the absence of values, or nothingness.

- **Operators:** Operators are specialized symbols that perform arithmetic or logical operations. The operation in the variable is applied using operands. The operators can be:
    - Arithmetic operators (+, -, * /, %, **, //)
    - Bitwise operators (&, ^, |)
    - Shift operators (<<, >>)
    - Identity operators (is, is not)
    - Relational operators (>, <, >=, <=, ==, !=)
    - Logical operators (and, or)
    - Assignment operator (=)
    - Membership operators (in, not in)
    - Arithmetic-assignment operators (/=, +=, -=, %=, **=, //=).
- **Punctuators:** The structures, statements, and expressions in Python are organized using these symbols known as punctuators. Several punctuators are in python [ ] { } ( ) @ -= += *= //= **== = , etc.

What is variable?

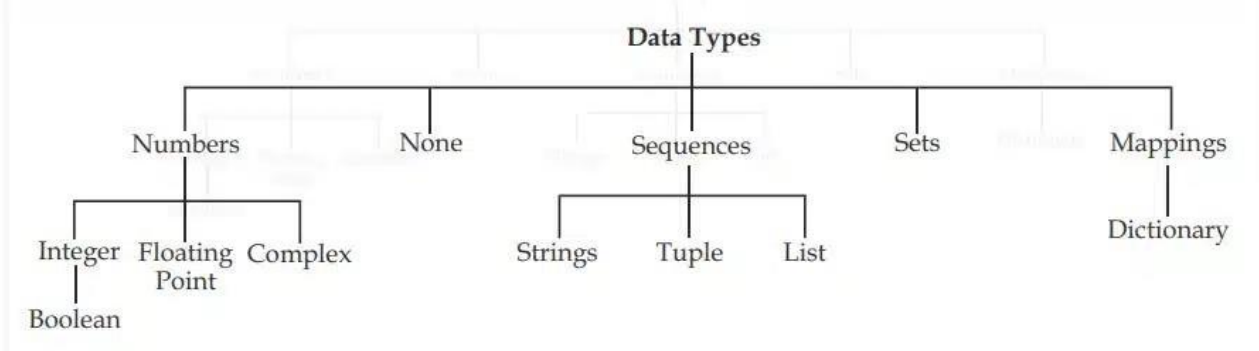
A variable is just like a container which helps to contain the values. It serves as an object or element that uses memory space, which can contain a value, variable number, alphanumeric or both. Example, name = “Python”

Use of comments in python

The statement ignored by the Python interpreter during the execution is known as a comment. The comment starts with a hash symbol (#) in Python. It helps to add remarks in the source code.

Data types in Python

In Python, data types define the type of data that a variable can store. The following chart shows the different types of data types in Python.



Data Types in Python			
Type	**Data Type**	**Description**	**Example**
Number	Int	Integer number	-10, -4, 0, 120, 1034
	Float	Real or floating point number	-3.08, 7.0, 23.41
	Complex	Complex number	3 + 4j, 2 – 2j
None	None is a special data type with a single value.		myVar = None
Sequence	String	Set of characters	"hello"
	List	Set of item separated by comma	[1, 2, 3, 'a']
	Tuple	Set of items separated by comma enclosed in (), it is not editable	(1, 'a', 2, 3)
Sets	Unordered data type, separated by commas and enclosed in curly brackets { }.		{10,20,3.14,"New Delhi"}
Mapping	Dictionary	Unordered data type, Holds data items in key-value pairs	{'a':1, 'b':2}

Operators in python

An operator is used to perform a specific mathematical or logical operation on values. The values that the operators work on are called operands. For example, in the expression 10 + num, the 10 is a value, the num is a variable, and the + (plus) sign is an operator.

- Arithmetic operators (+, -, * /, %, **, //)
- Bitwise operators (&, ^, |)
- Shift operators (<<, >>)
- Identity operators (is, is not)
- Relational operators (>, <, >=, <=, ==, !=)
- Logical operators (and, or)
- Assignment operator (=)
- Membership operators (in, not in)
- Arithmetic-assignment operators (/=, +=, -=, %=, **=, //=).

Expressions in Python

Expressions are representations of value, an expression is defined as a combination of constants, variables, and operators. A value or a standalone variable is also considered as an expression but a standalone operator is not an expression. Some examples of valid expressions are:

Expressions	
100	3.0 + 3.14
num	23/3 -5 * 7(14 -2)
num – 20.4	"Global" + "Citizen"

Statement in Python

In Python, a statement is a unit of code that the Python interpreter can execute.

```
>>> x = 4                    #assignment statement
>>> cube = x ** 3       #assignment statement
>>> print (x, cube)      #print statement
4 64
```

Input and output in Python

The Python program needs to interact with the user to get some input data or information. This is done by using the input() function, and the print() function helps to give output on the screen.

a. Input Function

```
fname = input("Enter your first name: ")
 Enter your first name: Arnab
```

b. Output Function

```
fname = input("Enter your first name: ")
print(fname)
Enter your first name: Arnab
Arnab
```

Some more examples of Output function

Statement	Output
print(“Hello”)	Hello
print(10 * 2.5)	25.0
print(“I” + “love” + “my” + “country”)	Ilovemycountry
print(“I’m”, 16, “years old”)	I’m 16 years old

Type-conversion (explicit and implicit conversion)

Type conversion refers to converting one type of data to another type, and this data conversion can be done using two different ways:

- **Explicit conversion:** Explicit conversion done by the programmer manually using the cast operator.
- **Implicit conversion:** Implicit conversion is done automatically by the compiler; it is also known as coercion.

a. **Explicit conversion example**

```
num1 = 11
num2 = 2

print(num1/num2)                  #output: 5.5
print(int(num1/num2))           #output: 5
```

Function	Description
int(x)	Converts x to an integer
float(x)	Converts x to a floating-point number
str(x)	Converts x to a string representation

b. **Implicit conversion example**

```
num1 = 10                              #num1 is an integer
num2 = 20.0                           #num2 is a float
sum1 = num1 + num2        #sum1 is sum of a float and an integer
print(sum1)
print(type(sum1))
```

Debugging in Python

A programmer can make mistakes while writing a program; the process of debugging, which helps to identify and remove mistakes, it is also known as bugs or errors. Errors are categorized as:

- **Syntax errors:** Every program has its own rules, and if a mistake is made in the program like missing any punctuation, incorrect command, or mismatched parentheses or braces, it can generate a syntax error.
- **Logical errors:** A logical error occurs when a program runs without errors but generates incorrect output, for example, assigning a value to the wrong variable, etc.
- **Runtime errors:** A runtime error occurs during the execution of a program or after the code is compiled or interpreted. For example, subtraction of two variables that hold string values, etc.

Statement Flow Control

Control flow refers to the sequence in which a program's code is executed. Control flow statements manage the order in which code is executed and help to create a dynamic and flexible program.

Python has three types of control structures –

- **Sequential** – By default mode
- **Selection** – Used in decision making like if, switch etc.

- **Repetition** – It is used in looping or repeating a code multiple time

Conditional statements

The “IF-ELSE” conditionals help to check whether a condition is true or false; it is also known as a conditional statement. An “IF” statement executes a block of code only if the condition is true, while an “IF-ELSE” statement executes a block of code in both conditions, if the condition is true or false.

Syntax of IF condition –

```
if <conditional expression>:
     [statement 1]
     [statement 2]
```

Syntax of IF-ELSE condition –

```
if <conditional expression>:
     [statement 1]
     [statement 2]
else:
     [statement 1]
     [statement 2]
```

Nested IF statement

Nested if statements in Python allow the placing of one if statement inside another. Nested If is a powerful tool for building complex decision-making logic in your programs.

Syntax of Nested IF statement

```
if condition:
     if condition:
          [Statement]
     else:
          [Statement]
```

Looping Statement

In Python, looping statements are used to run a block of statements or code continuously for as many times as the user specifies. Python offers us two different forms of loops for

The For loop

A “for” loop allows a block of code to be executed repeatedly until a condition is met. A for loop is used when you want to execute code multiple times or used for iterating over sequences like lists and arrays.

Syntax of FOR loop –

```
for <variable> in <sequence> :
     statements_to_repeat
```

The range() based for loop

The range() function allows you to generate a sequence of numbers, which can be used in a for loop to iterate a specific number of times. It helps to control the flow of loops and can be used in several ways. By default range() function starts from 0.

Syntax –

```
range(stop)
range(start, stop)
range(start, stop, step)
```

The While loop

A while loop is a conditional loop that will repeat the instructions within itself as long as a conditional remain true.

Syntax –

```
while <logical expression> :
          loop-body
```

Jump Statement (break and continue)

Python offers two jump statement – break and continue – to be used within loops to jump out of loop-iterations.

a. The break Statement

A break statement is used to terminate the loop based on the condition; a break loop is generally associated with an if statement. This loop termination can be used in a for loop, a do loop or a while loop.

Example:

```
a = b = c = 0
for i in range(1, 11) :
     a = int(input("Enter number 1 :"))
     b = int(input("Enter number 2 :))
     if b == 0 :
          print("Division by zero error! Aborting")
          break
     else
          c=a/b
          print("Quotient = ", c)
print("program over!")
```

b. The continue statement

Unlike break statement, the continue statement forces the next iteration of the loop to take place, skipping any code in between.

Loop else statement

Python supports combining the else keyword with both the for and while loops. In a loop else statement, the else statement is used after the loop. The loop else statement will be executed once when the loop is completed without encountering a break statement.

Example,

```
for i in range(5):
    print(i)
    if i == 3:
        break
else:
    print("Loop completed without breaking.")
```

Nested Loops

A loop may contain another loop in its body. This form of a loop is called nested loop. But in a nested loop, the inner loop must terminate before the outer loop.

The following is an example of a nested loop :

```
for i in range(1,6) :
     for j in range(1, i) :
          print("*", end =' ')
     print()
```

String in Python

The combination of characters is known as a string. A string is a fundamental data type in Python; it is immutable. Immutable means once a string is created, its value cannot be changed. Strings are enclosed in either single quotes (‘ ‘), double quotes (” “), or triple quotes (“”” “”””). Triple quotes are basically used for multiple lines.

Basic Operation

Operation/ Method	Description	Example
Concatenation	Combine strings using + operator	“Hello ” + “World” (**Output:** Hello World)
Repetition	Repeat strings using * operator	“Hi ” * 3 (**Output:** Hi Hi Hi)
Membership	Check substring presence using in or not in	“a” in “apple” (**Output:** True)
Slicing	Extract parts of the string using index ranges	“Hello”[0:4] (**Output:** Hell)

List in Python

In Python, Multiple values (example, Number, Character, Date etc.) can be stored in a single variable by using lists., a list is an ordered sequence of elements that can be changed or modified. A list's items are any elements or values that are contained within it. Lists are defined by having values inside square brackets [] just as strings are defined by characters inside quotations.

Example 1,

```
>>> list1 = [2,4,6,8,10,12]
>>>  print(list1)
[2, 4, 6, 8, 10, 12]
```

Tuples in Python

An ordered collection of components of various data kinds, such as integer, float, string, list, is known as a tuple. A tuple's components are denoted by parenthesis (round brackets) and commas. A tuple's elements can be retrieved using index values beginning at 0 just like list and string members can.

Example 1,

```
>>> tuple1 = (1,2,3,4,5)
>>> tuple1
(1, 2, 3, 4, 5)
```

Example 2,

```
>>> tuple2 =('Economics',87,'Accountancy',89.6)
>>> tuple2
('Economics', 87, 'Accountancy', 89.6
```

Dictionary in Python

Maps include the data type dictionary. A collection of keys and a set of values are mapped in this situation. Items are keys and values pairs. Consecutive entries are separated by commas, and a colon (:) separates a key from its value. Dictionary entries are unordered; thus, we might not receive the data in the same order that we entered it when we first placed it in the dictionary.

Example,

```
>>> dict3 = {'Mohan':95,'Ram':89,'Suhel':92,'Sangeeta':85}
>>> dict3
{'Mohan': 95, 'Ram': 89, 'Suhel': 92,'Sangeeta': 85}
```