

Python Fundamentals (Part1)

Concepts : Output/ Input, Variables, Data Types & Operators

Output

The first & the easiest thing that we can probably do in Python is printing (or outputting) something on our screen and we can perform this simple task using a special function, called the `print()` function.

What exactly is a function? We'll learn about it in the next chapter but for now we can imagine it to be a specialized helper that performs a specific task. So, `print()` is our little helper that helps us display something on the screen(the console). We can use it to show information, results of calculations or even format our text.

Syntax:

```
print("hello world")  
# output will be 'Hello World'
```

Anything quotes("") or double-quotes("") is called a string & gets printed as it is.

We can also print numerical values:

```
print(3.14)      # output: 3.14  
print(2 + 3)     # output: 5
```

Each print statement displays output on different lines but we can also display multiple values on same line.

```
print("hello world", "from PRIME")
```

Python Character Set

A Character set is a collection of characters, numbers, symbols that our language recognizes & we can use. Following are the common characters we'll be frequently using in Python:

1. English letters - 'A' to 'z' and 'a' to 'z'
2. Digits - 0 to 9
3. Special symbols - +, -, *, /, %, etc.
4. Whitespaces & Escape characters - blank space (), tab space (\t), newline (\n) etc.
5. All ASCII & Unicode characters as part of data & literals

Variables

A **variable** is like a **container** that stores **data** or **values** in our program. We can think of it as a labeled box where we can place something in, change it, or use it later.

There are variables in Python:

1. Have names called identifiers.
2. Can store data of any type (numbers, text etc.)

Some **examples**:

```
name = "Shradha"  
age = 30  
PI = 3.14  
word = "artificial intelligence"  
  
print ("value of PI =", PI)
```

Variable Naming(identifier) Rules

1. Name must start with a **letter** (a–z, A–Z) or **underscore** `_`.
2. Can contain **letters, numbers, or underscores** after the first character.
3. Cannot use **Python keywords** (like `if`, `for`, `class`) as variable names.

Let's look at some invalid variable names:

```
2name = "Bob"    # Starts with a number  
class = 10       # 'class' is a keyword
```



Important Note

1. Python is **dynamically typed**. So when creating variables we don't need to declare type explicitly.
2. Python is a **case-sensitive** i.e. variable `age` & `Age` are different.
3. **Indentation** in Python is important & the general standard is 4 spaces per indentation level.
4. **Keywords** are reserved words in Python & their meanings are predefined by the language.

Examples - `if`, `for`, `class`, `while`, `else`, `in`, `def` etc.

Data Types

A **data type** defines the **kind of value a variable can hold**. Python has several built-in data types.

Let's have a look at the simplest ones to start with:

1. `int` - integer values (whole numbers)
2. `float` - floating-point values (decimal numbers)
3. `str` - strings (sequence of characters like words & sentences)
4. `bool` - boolean values (`True` / `False`)
5. `None` - special constant representing no value / null

We also have a lot of other data types like `complex`, `list`, `tuple`, `set` etc. that we'll cover in coming chapters. To check the variable type we can use the `type()` function.

```
x = 10
print(type(x))          # <class 'int'>

PI = 3.14
print(type(PI))         # <class 'float'>

name = "shradha"
print(type(name))       # <class 'str'>

isTeacher = True
print(type(isAdult))    # <class 'bool'>

empty_var = None
print(type(empty_var))  # <class 'NoneType'>
```

Type Conversion & Type Casting

Type conversion is when we convert(cast) variables from one type to another. It can happen in 2 ways:

1. Type conversion - Implicit, done automatically by Python

```
a = 5
b = 3.0
print(a + b) # Python converts ans in float by default
```

2. Type Casting - Explicitly, done by the programmer

```
x = 10          # int
y = float(x)    # convert to float
z = str(x)      # convert to string
```

`int()` , `float()` , `str()` , `bool()` , `list()` , `tuple()` are common type conversion functions.

Operators

An **operator** is a symbol that tells Python to perform a **specific computation** on one or more values (called operands).

Example:

```
print(1 + 3) # '+' is an Operator & (1, 3) are operands
```

There are several types of operators in Python. Let's start by understanding some of the most common ones:

1. Arithmetic Operators - used to perform math operations.

Operators - `+` , `-` , `*` , `/` , `%` , `**`

```
a = 5
b = 10

print(a + b)    # Addition
print(a - b)    # Subtraction
print(a * b)    # Multiplication
print(a / b)    # Division
print(a % b)    # Modulo - remainder
print(a ** b)   # Power
```

2. Relational Operators - used to compare values.

Operators - `==`, `!=`, `>`, `>=`, `<`, `<=`

```
a = 10
b = 20

# Equal to
print("a == b:", a == b) # False

# Not equal to
print("a != b:", a != b) # True

# Greater than
print("a > b:", a > b) # False

# Less than
print("a < b:", a < b) # True

# Greater than or equal to
print("a >= b:", a >= b) # False

# Less than or equal to
print("a <= b:", a <= b) # True
```

3. Assignment Operators - used to assign values to variables.

Operators - `=`, `+=`, `-=`, `*=`, `/=`, `%=`, `**`

```
# Simple assignment
x = 10
print("x =", x) # 10

x += 5 # equivalent to x = x + 5
print("x += 5 ->", x) # 15

x -= 3 # equivalent to x = x - 3
print("x -= 3 ->", x) # 12

x *= 2 # equivalent to x = x * 2
print("x *= 2 ->", x) # 24

x /= 4 # equivalent to x = x / 4
print("x /= 4 ->", x) # 6.0

x %= 4 # equivalent to x = x % 4
print("x %= 4 ->", x) # 2.0

x **= 3 # equivalent to x = x ** 3
print("x **= 3 ->", x) # 8.0
```

4. Logical Operators - used to combine boolean values.

Operators - `and`, `or`, `not`

val1	val2	<code>and</code>
T	T	T
T	F	F
F	T	F
F	F	F

val1	val2	<code>or</code>
T	T	T
T	F	T
F	T	T
F	F	F

val1	<code>not</code>
T	F
F	T

```
x = 10
y = 20
z = 5

# AND
print(x > z and y > x) # True

# OR
print(x > y or y > z) # True


# NOT
print(not (x > y)) # True
```

We'll cover more operator types like membership operators in future chapters.

Operator Precedence

Operators have a **priority**; it means there is a specific order in which operations are performed when we have multiple operators appearing in the same expression. The order is as follows:

- `()` - Parentheses (highest priority)
- `**` - Exponent
- `+x`, `-x`, `~x` - Unary operators
- `/` `//` `%` - Multiplication, division, floor, modulus
- `+` `-` - Addition, subtraction
- `<<` `>>` - Bitwise shifts
- `&` - Bitwise AND
- `^` - Bitwise XOR
- `|` - Bitwise OR
- Comparison operators - `<`, `<=`, `>`, `>=`, `!=`, `==`
- `not`
- `and`
- `or`
- Assignment operators - `=`, `+=`, `-=` ...

 Note - Unary operators are operators that **works on a single operand** (a single value or variable) to perform an operation. These are like `not`, `+` (unary plus), `-` (unary minus) etc. `+x` is used to indicate positive value & `-x` to indicate negative value.

```
x = 5
y = -x # unary minus
print(x) # 5
print(y) # -5
```

We don't generally use `+x` as numbers are positive by default.

Input

We use the `input()` function to take input from user while the program is running. Whatever the user types in input prompt is returned as a string.

Syntax:

```
name = input("enter name: ")
print(name)
print(type(name))    # type is string
```

We can use type conversion functions to convert the type of input.

```
# program to add 2 numbers entered by the user
a = int(input("enter 1st num: "))
b = int(input("enter 2nd num: "))

print(a + b)
```

Let's summarize all the concepts we learnt in this chapter into a simple practice problem.

Write a program that takes input of 2 numbers (`a` & `b`) and print the average.

```
a = int(input("enter 1st num: ")) # 5
b = int(input("enter 2nd num: ")) # 10

avg = (a + b) / 2
print("average = ", avg) # 7.5
```

| *Keep Learning & Keep Exploring!*