Data Types in Python

In programming languages, every value or data has an associated type to it known as data type. Some commonly used data types.

String: A String is a stream of characters enclosed within quotes.

```
"Hello World!"
1234
```

Integer: All the numbers (positive, negative and zero) without any fractional part come under Integers.

```
...-3, -2, -1, 0, 1, 2, 3,...
```

Float: Any number with a decimal point.

```
24.3, 345.210, -321.86
```

Boolean: In a general sense, anything that can take one of two possible values is considered a Boolean. As per the Python Syntax, `True` and `False` are considered as Boolean values.

```
True, False
```

Conditional Statements

Conditional Statement: Conditional Statement allows you to execute a block of code only when a specific condition is `True`.

```
if True:
    print("If Block")
    print("Inside If")
```

```
# Output is:
If Block
Inside If
```

If - Else Statement: When the `If - Else` conditional statement is used, the Else block of code executes if the condition is `False`.

```
a = int(input()) # -1
if a > 0:
    print("Positive")
else:
    print("Not Positive")
```

```
# Output is:
Not Positive
```

Nested Conditions: The conditional block inside another if/else conditional block is called as a nested conditional block.

```
if Condition A:
    if Condition B:
        block of code
```

```
else:
    block of code
```

```
if Condition A:
    block of code
else:
    if Condition B:
        block of code
```

Elif Statement: Use the elif statement to have multiple conditional statements between if and else. The elif statement is optional.

```
if Condition A:
    block of code
elif Condition B:
    block of code
else:
    block of code
```

Identation:

1. Space(s) in front of the conditional block is called indentation.
2. Indentation(spacing) is used to identify the Conditional Blocks.
3. Standard practice is to use four spaces for indentation.

Strings - working with strings

String Concatenation: Joining strings together is called string concatenation.

```
a = "Hello" + " " + "World"
print(a) # Hello World
```

String Repetition: * operator is used for repeating strings any number of times as required.

```
a = "$" * 10
print(a) # $$$$$$$$$$
```

Length of String: `len()` returns the number of characters in a given string.

```
username = input() # Ravi
length = len(username)
print(length) # 4
```

String Indexing: We can access an individual character in a string using their positions (which start from 0) . These positions are also called *index*.

```
username = "Ravi"
first_letter = username[0]
print(first_letter) # R
```

String Slicing: Obtaining a part of a string is called string slicing. Start from the *start_index* and stops at the *end_index*. (end_index is not included in the slice).

```
message = "Hi Ravi"
part = message[3:7]
print(part) # Ravi
```

Slicing to End: If *end_index* is not specified, slicing stops at the end of the string.

```
message = "Hi Ravi"
part = message[3:]
print(part) # Ravi
```

Slicing from Start: If the *start_index* is not specified, the slicing starts from the index `0`.

```
message = "Hi Ravi"
part = message[:2]
print(part) # Hi
```

Negative Indexing: Use negative indexes to start the slice from the end of the string.

```
b = "Hello, World!"
print(b[-5:-2]) # orl
```

Reversing String: Reverse the given string using the extended slice operator.

```
txt = "Hello World"
txt = txt[::-1]
print(txt) # dlroW olleH
```

Membership check-in strings:

in: By using the `in` operator, one can determine if a value is present in a sequence or not.

```
language = "Python"
result = "P" in language
print(result) # True
```

not in: By using the, `not in` operator, one can determine if a value is not present in a sequence or not.

```
language = "Python"
result = "P" not in language
print(result) # False
```

Calculations in Python

Addition: Addition is denoted by `+` sign.

```
print(2 + 5)  # 7
print(1 + 1.5) # 2.5
```

Subtraction: Subtraction is denoted by `-` sign.

```
print(5 - 2) # 3
```

Multiplication: Multiplication is denoted by `*` sign.

```
print(2 * 5) # 10
print(5 * 0.5) # 2.5
```

Division: Division is denoted by `/` sign.

```
print(80 / 5) # 16.0
```

Modulus: To find the remainder, we use the Modulus operator `%`.

```
print(7 % 2) # 1
```

Exponent: To find `a` power `b`, we use Exponent Operator `**`.

```
print(7 ** 2) # 49
```

Floor division: To find an integral part of the quotient we use Floor Division Operator `//`.

```
print(13 // 5) # 2
```

Input and Output Basics

Take Input From User: `input()` allows flexibility to take input from the user. Reads a line of input as a string.

```
username = input() # Ajay
```

Printing the Output: `print()` function prints the message to the screen or any other standard output device.

```
print(username) # Ajay
```

Comments: Comment starts with a hash `#` . It can be written in its own line next to a statement of code.

```
# This is a comment
```

String Methods

Name	Syntax	Usage
`isdigit()`	`str.isdigit()`	Gives True if all the characters are digits. Otherwise, False.
`strip()`	`str.strip()`	Removes all the leading and trailing spaces from a string.
`strip()` with separator	`str.strip(separator)`	We can also specify separator(string) that need to be removed.
`replace()`	`str.replace(old, new)`	Gives a new string after replacing all the occurrences of the old substring with the new substring.
`startswith()`	`str_var.startswith(value)`	Gives True if the string starts with the specified value. Otherwise, False.
`endswith()`	`str.endswith(value)`	Gives True if the string ends with the specified value. Otherwise, False.
`upper()`	`str.upper()`	Gives a new string by converting each character of the given string to uppercase.

Name	Syntax	Usage
`lower()`	`str.lower()`	Gives a new string by converting each character of the given string to lowercase.
`split()`	`str.split()`	The split() method splits a string into a list.
`split()` with separator	`str.split(separator, maxsplit)`	Specifies the separator to use when splitting the string. By default any whitespace is a separator.
`join()`	`str.join(iterable)`	The join() method takes all items in an iterable and joins them into one string.

String Formatting: String Formatting simplifies the concatenation. It increases the readability of code and type conversion is not required.

Add Placeholders: Add placeholders {} where the string needs to be formatted.

```
name = "Raju"
age = 10
msg = "Hi {}. You are {} years old."
print(msg.format(name, age)) # Hi Raju. You are 10 years old.
```

Numbering Placeholders: Numbering placeholders, will fill values according to the position of arguments.

```
name = input() # Raju
age = int(input()) # 10
msg = "Hi {1}. You are {0} years old."
print(msg.format(name, age)) # Hi 10. You are Raju years old.
```

Naming Placeholder: Naming placeholders will fill values according to the keyword arguments.

```
name = input() # Raju
age = int(input()) # 10
msg = "Hi {name}. You are {age} years old."
print(msg.format(age=age, name=name)) # Hi Raju. You are 10 years old.
```

Relational & Logical Operators

Relational Operators are used to comparing values. Gives `True` or `False` as the result of a comparison.

Operator	Name	Example	Output
>	Is greater than	print(2 > 1)	True
<	Is less than	print(5 < 10)	True
==	Is equal to	print(3 == 4)	False
<=	Is less than or equal to	print(2 <= 1)	False
>=	Is greater than or equal to	print(2 >= 1)	True
!=	Is not equal to	print(2 != 1)	True

Logical operators are used to performing logical operations on Boolean values. Gives `True` or `False` as a result.

Name	Code	Output
and	print((5 < 10) and (1 < 2))	True
or	print((5 < 10) or (2 < 2))	True
not	print(not (2 < 3))	False

Logical Operators Truth Table:

A	B	A and B
True	True	True
True	False	False
False	False	False
False	True	False

A	B	A or B
True	True	True
True	False	True
False	False	False
False	True	True

A	Not A
True	False
False	True