NUMBER DATA TYPE

Numeric Types in Python

In Python, number data types are used to store numeric values. There are four different numerical types in Python:

- **int (plain integers):** this one is pretty standard -- plain integers are just positive or negative whole numbers.
- **long (long integers):** long integers are integers of infinite size. They look just like plain integers except they're followed by the letter "L" (ex: 150L).
- **float (floating point real values):** floats represent real numbers, but are written with decimal points (or scientific notaion) to divide the whole number into fractional parts.
- **complex (complex numbers):** represented by the formula a + bJ, where a and b are floats, and J is the square root of -1 (the result of which is an imaginary number). Complex numbers are used sparingly in Python.

Integers

There are three types of integer values supported by Python- Binary, Octal, and Hexadecimal.

```
In [1]: print(123123123123123123123123123123123123123123123123 + 1)
123123123123123123123123123123123123123123123124

In [2]: print(10)
10

In [3]: print(0o10)
8

In [4]: print(0x10)
16

In [5]: print(0b10)
2
```

Number system prefix for Python numbers

Numbers we deal with everyday are decimal (base 10) number system. But computer programmers (generally embedded programmer) need to work with binary (base 2), hexadecimal (base 16) and octal (base 8) number systems.

Number System	Prefix
Binary	'0b' or '0B'
Octal	'0o' or '0O'
Hexadecimal	'0x' or '0X'

Floating-Point Numbers

- The float type in Python designates a floating-point number. float values are specified with a decimal point. Optionally, the character e or E followed by a positive or negative integer may be appended to specify scientific notation:

Floating-Point Numbers

```
In [7]: 4.2
Out[7]: 4.2

In [8]: type(4.2)
Out[8]: float

In [9]: 4.
Out[9]: 4.0

In [10]: .2
Out[10]: 0.2

In [11]: float(2)
Out[11]: 2.0

In [12]: .4e7
Out[12]: 4000000.0

In [ ]:
```

Complex Numbers

- Complex numbers are specified as <real part>+<imaginary part>j. For example:

```
In [13]: 2+3j
Out[13]: (2+3j)

In [14]: type(2+3j)
Out[14]: complex

In [ ]:
```

Strings

- Strings are sequences of character data. The string type in Python is called str.
- String literals may be delimited using either single or double quotes.
- All the characters between the opening delimiter and matching closing delimiter are part of the string.

Strings

```
In [15]: print("I am a string.")
         I am a string.

In [16]: type("I am a string.")
Out[16]: str

In [17]: print('I am too.')
         I am too.

In [18]: type('I am too.')
Out[18]: str

In [19]: print("This string contains a single quote (') character.")
         This string contains a single quote (') character.

In [20]: print('This string contains a double quote (") character.')
         This string contains a double quote (") character.
```

Escape Sequences in Strings

Sometimes, you want Python to interpret a character or sequence of characters within a string differently. This may occur in one of two ways:

- You may want to suppress the special interpretation that certain characters are usually given within a string.
- You may want to apply special interpretation to characters in a string which would normally be taken literally.

Escape Sequences in Strings

```
In [23]: print('This string contains a single quote (') character.')
```

```
  File "<ipython-input-23-4fb72c6c5730>", line 1
    print('This string contains a single quote (') character.')
                                                  ^
SyntaxError: invalid syntax
```

```
In [21]: print('This string contains a single quote (\') character.')
```

```
This string contains a single quote (') character.
```

```
In [ ]:
```

Escape Sequences in Strings

The following is a table of escape sequences which cause Python to suppress the usual special interpretation of a character in a string:

Escape Sequence	Usual Interpretation of Character(s) After Backslash	"Escaped" Interpretation
`\'`	Terminates string with single quote opening delimiter	Literal single quote (') character
`\"`	Terminates string with double quote opening delimiter	Literal double quote (") character
`\newline`	Terminates input line	Newline is ignored
`\\`	Introduces escape sequence	Literal backslash (\) character

Escape Sequences in Strings

Here is a list of escape sequences that cause Python to apply special meaning instead of interpreting literally:

Escape Sequence	"Escaped" Interpretation
`\a`	ASCII Bell (BEL) character
`\b`	ASCII Backspace (BS) character
`\f`	ASCII Formfeed (FF) character
`\n`	ASCII Linefeed (LF) character
`\N{<name>}`	Character from Unicode database with given `<name>`
`\r`	ASCII Carriage Return (CR) character
`\t`	ASCII Horizontal Tab (TAB) character
`\uxxxx`	Unicode character with 16-bit hex value `xxxx`
`\Uxxxxxxxx`	Unicode character with 32-bit hex value `xxxxxxxx`
`\v`	ASCII Vertical Tab (VT) character
`\oxx`	Character with octal value `xx`
`\xhh`	Character with hex value `hh`

Operators

Arithmetic Operator

Operator	Description	Example
+ Addition	Adds values on either side of the operator.	a + b = 30
- Subtraction	Subtracts right hand operand from left hand operand.	a – b = -10
* Multiplication	Multiplies values on either side of the operator	a * b = 200
/ Division	Divides left hand operand by right hand operand	b / a = 2
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 0

Comparison Operators

Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	(a == b) is not true.
!= or <>	If values of two operands are not equal, then condition becomes true.	(a != b) is tru (a <>b) is true.
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.

Assignment Operators

Operator	Description	Example
=	Assigns values from right side operands to left side operand	c = a + b assigns value of a + b into c
+= Add AND	It adds right operand to the left operand and assign the result to left operand	c += a is equivalent to c = c + a
-= Subtract AND	It subtracts right operand from the left operand and assign the result to left operand	c -= a is equivalent to c = c - a
*= Multiply AND	It multiplies right operand with the left operand and assign the result to left operand	c *= a is equivalent to c = c * a
/= Divide AND	It divides left operand with the right operand and assign the result to left operand	c /= a is equivalent to c = c /a

Logical Operators

Operator	Description	Example
and Logical AND	If both the operands are true then condition becomes true.	(a and b) is true.
or Logical OR	If any of the two operands are non-zero then condition becomes true.	(a or b) is true.
not Logical NOT	Used to reverse the logical state of its operand.	Not(a and b) is false.

Input and Output

Input/Output

Functions like input() and print() are used for standard input and output operations respectively.

print()

```
print(*objects, sep=' ', end='\n')

print(1,2,3,4,sep='#',end='&')       # Output: 1#2#3#4&
```

input()

```
num = input('Enter a number: ')

Enter a number: 10

print(num)                           #Output: 10
```