For many of you, it is your very first programming course. Here we will give you a brief idea about how to solve problems by writing programs in Python.

Data types, Values and Variables

Data types

The data type is mainly the category of the data. There are basically 5 groups of main data types but there are various other types available in python.

1. Numeric type:
    a) Integer (int): Positive or negative whole numbers (without a fractional part).
    For example: 10,1000000,-3
    b) Floating-point (float): Any real numbers with “decimal” points or floating-point representation.
    For example: -3.1234, 100.3458998

2. Boolean type (bool): True and False.

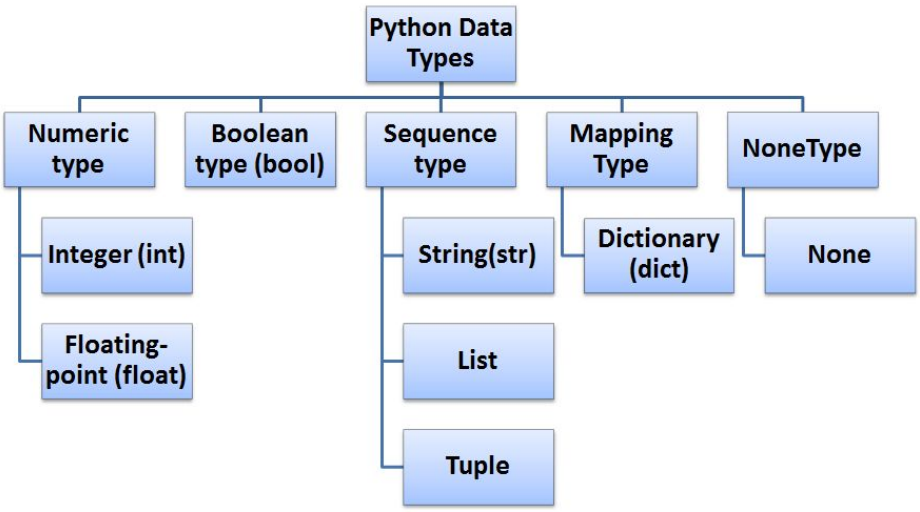
3. Sequence type: A sequence is an ordered collection of similar or different data types.

    a) String(str): It is a sequence of ordered characters (alphabets-lower case, upper case, numeric values, special symbols) represented with quotation marks (single('), double(''), triple(' ' ', '' '' '')).
    For example: using single quotes ('CSE110'), double quotes ("CSE110") or triple quotes ('''CSE110''' or """CSE110"""), "Hello CSE110 students".
    Note: Details will be discussed later.

    b) List: It is an ordered collection of elements where the elements are separated with a comma (,) and enclosed within square brackets []. The list can have elements with more than ode data types.
    For example: list with only integers [1, 2, 3] and a list with mixed data types [110, "CSE110", 12.4550, [], None]. Here, 110 is an integer, "CSE110" is a string, 12.4550 is a floating-point number, [] is a list, None is NoneType.
    Note: Details will be discussed later.

    c) Tuple (Note: This topic will be discussed later.)

4. Mapping Type: Dictionary (dict) (Note: will be discussed later.)
5.NoneType(None): It refers to a null value or no value at all. It's a special data type with a single value, None.
Note: Details will be discussed later.



Data type checking

Mainly for debugging (code correcting) purpose type() function is used. It returns the type of argument(object) passed as a parameter.

Example:

Code	Output
`type("Hello CSE110")`	`str`
`type(110)`	`int`
`type(123.456)`	`float`
`type(True)`	`bool`
`type(None)`	`NoneType`

`print()` function

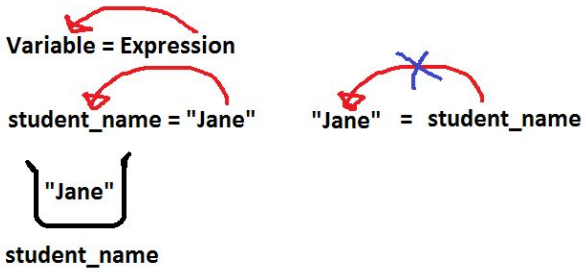
`the print()` function is used for printing the value of an expression written inside the parentheses (pair of an opening and closing first brackets). We shall discuss more about this after you have learned what is a function in python. In a command-line interpreter, you may not need the `print()` function, but when we write a set of code and want to see any text output on the screen, we use the `print()` function. For example:

Code	Output
`print("Hello CSE110 class")`	`Hello CSE110 class`

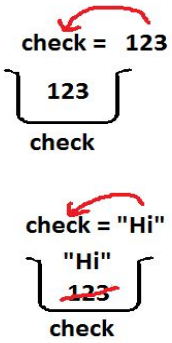
`print(123)`	`123`
`print(123.567)`	`123.567`
`print(type(123.456))`	`<class 'float'>`

Variable

Variables are containers for storing data values.



Here, student_name is a variable and it is holding or referring to the value “Jane”. Values are always stored from the right side of the assignment(=) to the variable on the left side. Unlike other languages, in python, we do not need to declare data types directly. The data type is decided by the interpreter during the run-time.



one variable can be assigned values of different data types. Here, in this example, first 123 was put in the container or variable named check. Then we replaced the value of the container from 123 to “Hi” which is a string. For example:

Code	Output

`check = 123`
`print(check)`
`print(type(check))`
`print("======")`
	`123`
	`<class 'int'>`
	`======`
	`Hi`

`check = "Hi"`
`print(check)`
`print(type(check))`
	`<class 'str'>`

Variable naming conventions

While naming the variables we need to follow certain rules and restrictions. These are called “Variable naming conventions”. By following these our codes become more understandable by the Human Readers (us). Few significant rules have been mentioned below.

Variable names-

- can have letters (A-Z and a-z), digits(0-9), and underscores (_).
- should maintain snake_casing. That means each word should be separated by underscores(_)
- cannot begin with digits
- cannot have whitespace and special signs (e.g.: +, -, !, @, $, #, %.)
- are case sensitive meaning ABC, Abc, abc are three different variables.
- have to be meaningful with the data stored in it.
- cannot be too lengthy and too general, need to have a balance
- should not be written with single alphabets. Minimum length should be 3.
- cannot be a reserved keyword for Python.

Python reserved keywords list							
and	del	from	not	while	assert	continue	def
as	elif	global	or	with	break	finally	for
else	if	pass	yield	except	import	lambda	is
print	class	exec	in	raise	return	try	

Detailed variable naming conventions can be found in the following
https://www.python.org/dev/peps/pep-0008/#prescriptive-naming-conventions

Basic operations and precedence

Now you know what data types are. A programming language uses 'operations' to manipulate the data stored in variables to achieve the desired results. Basic operations in Python can be divided into two parts: Unary (meaning it is done with one variable) and Binary (meaning it is done using two variables or one variable and a single value of data).
Note: You cannot do any operation with None type.

Let's explore each type of operation and understand what they do.

Unary operations

1. Unary + (plus): We use unary + (plus) operation by adding a '+' before a variable or data. It does not change the data. (Works with int, float, complex, and boolean. For booleans, True and False will be valued as 1 and 0 respectively.) For example:

Code	Output

`x = 5`
`print(x)`
`print(+x)`
	`5`
	`5`

2. Unary - (minus): We use unary - (minus) operation by adding a '-' before a variable or data. It produces the negative value of the input (equivalent to the multiplication with -1). (Works with int, float, complex, and boolean. For booleans, True and False will be valued as 1 and 0 respectively.) For example:

Code	Output

`x = 5`
`print(x)`
`print(-x)`
	`5`
	`-5`

3. Unary ~ (invert): We use unary - (invert) operation by adding a '~' before a variable or data. It produces a bitwise inverse of a given data. Simply, for any data x, a bitwise inverse is defined in python as -(x+1). (Works with int, and boolean. For booleans, True and False will be valued as 1 and 0 respectively.) For example:

Code	Output

`x = 5`
`print(x)`
`print(~x)`
	`5`
	`-6`

Binary operation:

Operators are symbols that represent any kind of computation such as addition, subtraction, and etc. The values or the variables the operator works on are called Operands.

1 + 2

here, 1 and 2 are operands, and + is the operator computing addition.

1. Arithmetic operation
    a. + (addition)
    b. - (subtraction)
    c. * (multiplication)

Code	Output
`print(2+3)`	`5`
`print(2-30)`	`-28`
`print(2*30)`	`60`
`print(2.456789*30)`	`73.70367`

    d. / (division)

`Division of numbers(float, int) yields a float`

Code	Output
`print(4/2)` `print(9/2)`	`2.0` `4.5`
`print(-5/2)`	`-2.5`
`print(30/4)`	`7.5`

    e. // (floor division)

`Floor division of integers results in an integer.`
`If one of the operands is float, then the floor division`

results in a float.

Code	Output
print(30//4)	7 (here, from the previous example, we can see, 30 divided by 4 yields 7.5. But while using the floor division (//), it basically discards the part after the decimal point and returns a whole number by flooring the value to the nearest integer number. Or we can say that it returns the quotient value after dividing)
print(10.5//2.5) print(10.5//2) print(10//2.5)	4.0 5.0 4.0 (If one of the operands is float, then the floor division results in a float.)
print(-5//2)	-3 Here, -5 divide by 2 gives -2.5 . Since we have used floor divide, it will take it to the nearest (lower) integer, which is -3.

f. % (modulus)

g.

(0) For positive values:

z = x%y

Here, x is the dividend, y is the divisor and z is the remainder.

Code	Output
print(30%4)	2 (here, the percentage sign is called "modulus operator" and it basically returns the remainder by dividing the first number with the second number. 30 divide by 4 yields quotient,4 and remainder 2. So the final output is 2)

(1) For negative values:

First option (For negative values**):**

z = x - y*(x//y)

x%y, then, the modulo operator always yields a result with the same sign as its second operand(the divisor) or zero

Code	Output
print(-5%2)	1 Here, x = -5, y = 2 z = -5 - 2*(-5//2) = - 5 - 2*(-3) = - 5 +6 = 1
print(-5.5%2)	0.5 Here, x = -5.5 , y = 2 z = -5.5 - 2*(-5.5//2) = - 5.5 - 2*(-3) = - 5.5 +6 = 0.5
print(5%-2)	-1 Here, x = 5, y = -2 z = 5 - (-2)*(5//-2) = 5 - (-2)*(-3) = 5 - (6) = -1
print(5.5%-2)	Here, x = 5.5, y = -2 z = 5.5 - (-2)*(5.5//-2) = 5.5 - (-2)*(-3.0) = 5.5 - (6.0) = - 0.5

Second option (For negative values**):**

x%y = (a+b)mod y = [(a mod y)+(b mod y)]mod y

Here, the x, dividend is broken into two parts such that, one part is always positive between a and b.

Code	Output
print(-5%2)	1 Here, -5%2 = (3-8)%2 = [3%2 + (-8%2)]%2 = (1+0)%2=1
print(-5.5%2)	0.5 Here, -5.5%2 = (2.5-8)%2 = [2.5%2 + (-8%2)]%2 = (0.5+0)%2 =0.5
print(5%-2)	-1 Here, 5%-2 = (4+1)%-2 = [(4%-2) + (1%-2)]%2 = [0+ -1]%2 = -1
print(5.5%-2)	-0.5 Here, 5.5%-2 = (4+1.5)%-2 = [(4%-2) + (1.5%-2)]%-2 = [0+ -1.5]%-2 = (-1.5)%-2= (-2+0.5)%-2 =[(-2%-2) + 0.5%-2]= 0+ (-0.5) = -0.5

h. ** (Exponentiation)

Basically it's the power operator (X**Y)= x^Y

Code	Output
print(2**4)	16
print(-3**2)	-9 (Here, $-3^2 = -9$)
print((-3)**2)	9 (Here, $(-3)^2 = 9$)
print(4**-2)	0.0625 Here, $4^{-2} = \frac{1}{4^2} = \frac{1}{16}$

Details can be found in the following link:
https://docs.python.org/3/reference/expressions.html#binary-arithmetic-operations

1. Assignment operator
    a. = (assign): It is used for putting value from the right side of the equal sign(=) to a variable on the left side of the equal sign(=).
    For example: number = 123.

    b. Compound Assignment Operators:
        i. += (add and assign)
        ii. -= (subtract and assign)
        iii. *= (multiply and assign)
        iv. /= (divide and assign)
        v. %= (modulus and assign)
        vi. **= (exponent and assign)
        vii. //= (floor division and assign)

Compound Assignment Operators	Example	
	Short-form	full form
+=	`a += 7`	`a = a + 7`
-=	`b -= 2`	`b = b - 2`
*=	`c *= 3`	`c = c * 3`
/=	`d /= 9`	`d = d / 9`
%=	`e %= 2`	`e = e % 2`
**=	`f **= 4`	`f = f ** 4`
//=	`g //= 11`	`g = g // 11`

2. Logical operator
    a. and (logical AND)
    b. or (logical OR)
    c. not (Logical NOT)
    Note: Details will be discussed in the branching chapter.

3. Comparison or Relational operator
    a. == (equal)
    b. != (not equal)
    c. > (greater than)
    d. < (less than)

e. >= (greater than or equal)
f. <= (less than or equal)
Note: Details will be discussed in the branching chapter.

4. Membership Operator
   a. in: Returns True if the first value is in the second. Otherwise, returns False.
   b. not in: Returns True if the first value is not in the second. Otherwise, returns False.

Example:

Code	Output
`print(4 in [1, 2, 3, 4, 5])`	`True`
`print("p" in "Hello")`	`False`
`print(100 not in [1, 2, 3, 4, 5])`	`True`
`print("p" not in "Hello")`	`True`

5. Identity Operators
Identity operators check whether two values are identical or not.

   a. is: Returns True if the first value is identical or the same as the second value. Otherwise, returns False.
   b. is not: Returns False if the first value is identical or the same as the second value. Otherwise, returns True.

Example:

Code	Output
`print("123" is 123)`	`False`
`print("123" is "123")`	`True`
`print(5.123 is 5.1234)`	`False`
`print("123" is not 123.00)`	`True`
`print("123" is not "123")`	`False`

7. Bitwise Operators [Note: will be covered in future courses]

a. & (Bitwise and)
b. | (Bitwise or)
c. ^ (Bitwise xor)
d. ~ (Bitwise 1's complement)
e. << (Bitwise left-shift)
f. >> (Bitwise right-shift)

Compound expression:

When we write an expression with two or more operations, it is called a compound expression. For example, we may write `7+9*3` where we have both addition and multiplication operations in a single expression. Determining the result of a compound expression is a little tricky. We need to think about what operation will be executed first. To determine that the computer follows Operator precedence, a set of rules that dictates the computer should be done first. For our example, `7+9*3` the multiplication operation will have higher precedence and will be executed first. So the program will first calculate `9*3` which results in 27. Then it will calculate 7+27, which will result in 34.

Try the following examples and see what results in it shows:

5*3+2-1*2

1+7/7*7

Operator precedence:

In the tale below precedence has been shown in descending order. Highest precedence at the top, lowest at the bottom. Operators in the same precedence are evaluated from left to right.

Precedence	Operator	Meaning
Highest	`()`	Parentheses (grouping)
	f`(args...)`	Function call
	x`[index:index]`	Slicing
	x[index]	Subscription
	x.attribute	Attribute reference
	`**`	Exponentiation
	`~`*x*	Bitwise not
	`+`*x*`, -`*x*	Positive, negative
	`*, /, %, //`	Multiplication, division, modulus, floor division

	+, -	Addition, subtraction
	<<, >>	Bitwise shifts
	&	Bitwise AND
	^	Bitwise XOR
	|	Bitwise OR
	in, not in, is, is not, <, <=, >, >=, <>, !=, ==	Comparisons, membership, identity
	not *x*	Boolean NOT
	and	Boolean AND
	or	Boolean OR
Lowest	lambda	Lambda expression

Example:

Code	Explanation and output
print(True or False and True)	here, "and" has precedence over "or" print(True or (False and True)) print(True or (False)) print(True or False) print(True) Output: True
print((True or False) and True)	here, parenthesis "() " has precedence over both "or" and "and" print((True or False) and True) print((True) and True) print(True and True) Output: True
print(9 - 5 // 2 * 13 + 2 ** 2)	here, Exponentiation (**) has the highest precedence. So, 2**2 will be executed first

	print(9 - 5 // 2 * 13 + 2 ** 2) in the next step, floor division(//), and multiplication(*) has the same precedence if the precedence is the same then, we need to execute from left to right. So, 5 // 2 print(9 - 5 // 2 * 13 + 4) print(9 - 2 * 13 + 4) then, * has the highest precedence. So 2 * 13 executes print(9 - 2 * 13 + 4) print(9 - 26 + 4) Here,- and + has the same precedence, so again left to right. print(9 - 26 + 4) print(-17 + 4) print(-13) Output: -13
print(1.5 * 10 // 2 + 8)	here, * and // has the same precedence, so execute from left to right. Execute 1.5 * 10 print(1.5 * 10 // 2 + 8) print(15.0 // 2 + 8) Then execute 15.0 // 2 print(15.0 // 2 + 8) print(7.0 + 8) print(7.0 + 8) print(15.0) Output: 15.0
print(-3 ** 2 * 2.0)	here, ** has precedence over *. print(-3 ** 2 * 2.0)

	print(-9 * 2.0) print(-9 * 2.0) print(-18.0) Output: -18.0
print(-3 ** (2 * 2.0))	here, () has precedence over **. print(-3 ** (2 * 2.0)) print(-3 ** (4.0)) print(-3 ** 4.0) print(-81.0) Output: -81.0

Type Conversion:

Data types can be modified in two ways: Implicit (when the compiler changes the type of a data for you) and Explicit (when you change the type of a data using type changing functions, it is called "*Type-casting*")

Note: What is the compiler will be explained later. Assume it to be like a teacher who checks your code for any kind of mistakes and shows the problems(errors) and sometimes does type conversion for you according to the computation need of a statement.

1. Implicit Type Conversion:
   If any of the operands are floating-point, then arithmetic operation yields a floating-point value. If the result was integer instead of floating-point, then removal of the fractional part would lead to the loss of information.

Operation	Result
5.0 -3	2.0
4.0/5	0.8
4/5.0	0.8
4*3.0	12.0
5//2.0	2.0
7.0%2	1.0

2. Explicit Type Conversion:
   Conversion among different data types are possible by using type conversion functions, though there are few restrictions. Python has several functions for this purpose among them these 3 are most used:
   a. str() : constructs a string from various data types such as strings, integer numbers and float-point numbers. For example:

Code	Output

`print(str(1))`
`print(type(str(1)))`
`print("=============")`
 	
`print(str(1.555))`
`print(type(str(1)))`
	`1`
	`<class 'str'>`
	`=============`
	`1.555`
	`<class 'str'>`

   b. int(): constructs an integer number from various data types such as strings (the input string has to consist of numbers without any decimal points, basically whole numbers), and float-point numbers (by rounding up to a whole number, basically it truncates the decimal part). For example:

Code	Output

`print(int("12"))`
`print(type(int("12")))`
`print("=============")`
 	
`print(int(12.34))`
`print(type(int(12.34)))`
	`12`
	`<class 'int'>`
	`=============`
	`12`
	`<class 'int'>`

`print(int("12.34"))`	`ValueError`
`print(int("12b"))`	`ValueError`

   c. float(): constructs a floating-point number from various data types such as integer numbers, and strings (the input string has to be a whole number or a floating point number). For example:

Code	Output

`print(float(12))`
`print(type(float(12)))`
`print("=============")`
	`12.0`
	`<class 'float'>`

print(float("34")) print(type(float("34"))) print("=============") print(float("34.56789")) print(type(float("34.56789"))) print("=============")	============= 34.0 <class 'float'> ============= 34.56789 <class 'float'> =============
print(float("34.5b789"))	ValueError
print(float("34b6"))	ValueError

Input

For taking input from the user directly, Python has a special built-in function, input(). It takes a String as a prompt argument and it is optional. It is used to display information or messages to the user regarding the input. For example, “Please enter a number” is a prompt. After typing the data/input in the designated inbox box provided by the IDE, the user needs to press the ENTER key, otherwise the program will be waiting for the user input indefinitely. The input() function converts it to a string and then returns the string to be assigned to the target variable.

Example: Taking name(str) and age(int) as input from user

```
name = input("Please enter your name: ")
print("Your name is " + name)
print("name data type: ", type(name))

age = input("Please enter your age: ")
print("Your age is " + age)
print("age data type: ", type(age))
```

Input	Output
Jane 20	Please enter your name: Jane Your name is Jane name data type: <class 'str'> Please enter your age: 20 Your age is 20 age data type: <class 'str'>

Here, from the example output, we can see that age has a String data type which was integer during the initial data input phase. This implicit data conversion has taken place in the input() function. No matter what data type we give as an input, this function always converts it to a string. So in order to get a number as an input, we need to use **explicit type conversion** on the input value.

Example: Taking a floating-point as an input

```
number = float(input("Please enter a floating-point number:"))
print("You have entered:"+ str (number))
print("Data type is",type(number))
```

Input	Output
12.75	Please enter a floating-point

	number:12.75 You have entered:12.75 Data type is <class 'float'>

Example: Taking integer as a input	
number = int(input("Please enter a number:")) print("You have entered: "+ str (number)) print("Data type is",type(number))	
Input	**Output**
13	Please enter a number:13 You have entered: 13 Data type is <class 'int'>

Style Guide for Python Code

For every programming language, there are few coding conventions followed by the coding community of that language. All those conventions or rules are stored in a collected document manner for the convenience of the coders, and it is called the “Style Guide” of that particular programming language. The provided link gives the style guidance for Python code comprising the standard library in the main Python distribution.

Python style guide link: https://www.python.org/dev/peps/pep-0008/