tutorial007-introToBasicPython

September 27, 2022

1 Introduction to basic Python variables and operators

1.1 Learning goals

- understand more details on the mechanics of coding
- basic types of variables in Python
- basic types of operators that can modify variables

1.2 Understanding the basic Python data types

In the previous tutorials, we have used code written to perform interesting operations. We loaded data, made plots, modified the plots by passing 'parameters' to 'functions', etc. This tutorial is about understanding the meaning of some of the foundamental operations in Python programming.

Most (if not all) programming works by assigning value to variables, and performing operations (calculations) that change the value of assigned to variables. In a certain way this is similar to what you have learned in math. For example:

- A variable say `a` is assigned a value say `3`.
- A function (or an operator) is used to modify the value of the variable `a`, say you might want to square the function (a square operation can be implemented as `a` *times* `a`).
- The result of the calculation will need to be stored in a variable different from `a`, say `b`, to be used later on, for other operations, or simply to report the results.

The above squaring operation can be streamlined as follow:

```
a = 3
b = a*a
```

Logic similar to the above is often used in programming. The first operation the one using the `=` is called **variable assigment**, a value is stored in a variable for later reuse. The second line above, the one containing `*` is an example of use of an **operator**, the symbol that implements the operation of multiplying one or more variables (the operator was `*` in our case.

Answer the following questions.

- Copy and paste hereafter two examples of *variable assigment* from `tutorial001` and `tutorial006`:
  - [x = np.linspace(1,20, 40)]
  - [myDataFromFile = pd.read_csv("datasets/006DataFile.csv")]

- Without yet knowing much about the types of operators, can you guess which one of the options below is not an example of an *operator*?
  - A. `*`
  - B. `/`
  - C. `+`
  - D. `-`
  - E. `A`

  [E. `A`]

We have seen variable assigments (and some types of operators) in the previous tutorials. So far, we have not yet described the basic operators and variables types that `Python` offers to work with. This is precisely what we will do below.

1.3 The basic variables types in Python

A distinction between variables types is needed because a computer needs to decide how much memeory to store for each variable. Memory depends not so much on the value of a variable (say if a variable is 2 or 100 or 1,000), but more on the type of variable (say if a variable is text, numbers, or if numbers have decimals or not). For example, decimals (such as 1.5) will require more space to be stored on your computer memory than simpler, entire numbers (such as 1 or 2).

Python provides four types of basic variable types.

Note that below we will use the python function `type()` to return the, ahem, type of each variable.

1.3.1 1. Integer (int)

These are used to describe variables with *whole* values, variables not requiring to store decimals.

[1]:
```
1 + 1 # 1 is an integer and the sum of two integers is an integer
type(1 + 1)
```

[1]:
```
int
```

[2]:
```
a = 4 # a variable can be assigned a whole number and the variable will be␣
 ↪defined as an integer
type(a)
```

[2]:
```
int
```

Answer the following questions.

- Insert three example (not used above) of `int` numbers: [6, -9, 8]

1.3.2 2. Floats

Conversely, floats are used to store variables and values that do need decimals to represent them.

Note. If you are interested to learn more about Floating Point numbers (Floats) Wikipedia has a nice Article about it: Floating Point Arithmetic

[3]:
```
c = 2.1
type(c)
```

[3]:
```
float
```

Answer the following questions.

- Insert three example (not used above) of `float` numbers: [5.5, -3.25, 8.99]

1.3.3 3. Complex

Complex data types store values such as the complex numbers in Math.

Note. We will not use these variables in our tutorials, and if you have never encountered complex numbers before, you should nto feel obliged to learn about them now. yet, chances are you might encoutner complex numbers if you were to become a serious data scientists.

[4]:
```
a = 1.5 + 0.5j
type(a)
```

[4]:
```
complex
```

note `j` is an existing variable in Python, preconfigures to serve the purpuse of allowing creating complex datatypes. The variable represents the imaginary unit.

The value of a complex variable comprises of a `real` and `imag` (short for imaginary) component. We can query the components using the `.` operator to index the component of the variable value we wish to recall:

[5]:
```
a.real # this is the real part of `a`
```

[5]:
```
1.5
```

[6]:
```
a.imag # this is the imaginary part of `a`
```

[6]:
```
0.5
```

Answer the following questions.

- Insert three example (not used above) of `complex` numbers: [3+2j, 10j-7, 2.1+9/2j]

1.3.4 Python will change your type!

Python is a dynamically typed language. Dynamically-typed means that we can define a variable, assign a value to it and we can then change the datatype associated with a variable. There are several built-in datatypes that are good to be aware of. This is good thing for us starting to work with the programming language because it makes the language less strict, easier to learn, although perhaps also more prone to potential mistakes and bugs. This is perhaps also one of the reasons why Python is so famous in data science and scientific programming.

When using a dynamically-typed language we need to be aware of it because it will be possible to combine some variables safely but not others. If two (or more) variables have the same type (i.e.,

they are both integer, floats or imaginary) they can be combined safely and their combination will maintain the original type. If two (or more) variables are not the same type, combining them will change the type of the variable to the *more complex one*.

Complexity here is a vague but easy-to-remmeber term. Variabless are stored differently on your computer, some need less memeory than others. `Integers` need less memory to be represented than `Floats` and `Complex` need more memory then either of them. Python is smart (i.e., dynamically typed) so it can make an educated guess to how to combine into an output numerical variables of different type, yet, in doing so it will need to change the type of the output variables. The simplest type will be dropped and the more complex, or memory greedy type will be returned as the type of the output.

For example:

[7]:
```
a = 1.5 + 0.5j # a is a complex type variable
type(a)
```

[7]:
```
complex
```

[8]:
```
b = 2 # b is defined as an integer type variable
type(b)
```

[8]:
```
int
```

[9]:
```
c = a + b # a is a complex type variable (the more complex type takes over)
type(c)
```

[9]:
```
complex
```

[10]:
```
a = 1.5 # a is a Float
type(a)
```

[10]:
```
float
```

[11]:
```
b = 2 # b is an int
type(2)
```

[11]:
```
int
```

[12]:
```
c = a + b  # c is a Float (the more complex type takes over)
type(c)
```

[12]:
```
float
```

Answer the following questions.

- Write an example of a `float` variable added to an `int` variable. Specify each variable name and show the type for each variable, as well as for the output variable. Write the code in the cell below.

[13]:
```
a = 2.45 # a is a float variable
type(a)
```

[13]:
```
float
```

[14]:
```
b = 18 # b is an int variable
type(b)
```

[14]:
```
int
```

[15]:
```
c = a + b # the output c is a float variable
type(c)
```

[15]:
```
float
```

Answer the following questions.

- Write an example of a `float` variable added to an `complex` variable. Specify each variable name and show the type for each variable, as well as for the output variable. Write the code in the cell below.

[16]:
```
a = 8.9 # a is a float variable
type(a)
```

[16]:
```
float
```

[17]:
```
b = 15j + 8 # b is a complex variable
type(b)
```

[17]:
```
complex
```

[18]:
```
c = a + b # the output c is a complex variable
type(c)
```

[18]:
```
complex
```

1.3.5 4. Booleans

In addition to numerical variables, Python can also represent other types of variables. Logical variables are stored in a dedicated (memory efficient) data type, called `bool` (short for boolean).

`Booleans` can take only one of two possible values `True` or `False`. Using boolean variables can make your code fast and efficient. But that is something you will worry about only in the future. For the moment, let's see how `bool` can be defined in Python.

[19]:
```
a = True
type(a)
```

[19]:
```
bool
```

[20]:
```
b = False
type(b)
```

[20]:
```
bool
```

Note `True` and `False` are not just words, but proper Python commands and must have an **Upper Case** first letter. Using a lowercase true returns an error. Actually, under the Python-hood, `True` and `False`, are defined as special versions of the `int 1` (`True`) and `0` (`False`). This is helpful to know because `True` and `False` behave as such in arithmetic contexts.

Answer the following questions.

- What happens if you define a new variable `a` using the word `false` (without uppercase)? [The result shows that "false" is not defined.]

A `bool` variable can be also created by requesting a logical operation, for example:

[21]:
```
a = 3 > 4
type(a)
```

[21]:
```
bool
```

Whereas the type of `a` is `bool`, its value is `False`, because `3` is not greater than `4`. Below we use the python function `print()` to return (print on screen) the content of the variable `a`:

[22]:
```
print(a)
```

```
False
```

The next example will create a logical (`bool`) variable with value set to `True`, this is because the result of the operation is actually true, `3` is less or equal than `3`.

[23]:
```
b = 3 <= 3
type(b)
```

[23]:
```
bool
```

[24]:
```
print(b)
```

```
True
```

Answer the following questions.

- What happens if you define a two variables `a` which is `bool` and `b` which is `int` and attempt to add them together? [Use the following cell to write code to implement the situation describe here.]

[25]:
```
a = True
type(a)
```

[25]:
```
bool
```

[26]:
```
b = 9
type(b)
```

[26]:
```
int
```

[27]:
```
a + b
```

[27]:
```
10
```

[When the boolean variable is True then the output is the integer + 1, and if the variable is False then the output is the integer + 0.]

1.3.6 5. Strings

Text is used very often in programming. Words might be needed to be dsiplayed say on top of a plot as a title or next to the axes of the plot. Text has a dedicated variable type in Python: `str` (short for string).

[28]:
```
a = 'hello'
type(a)
```

[28]:
```
str
```

Space is a character in Python (just like in a terminal!, oh my gosh thank you python).

[29]:
```
b = ' '
type(b)
```

[29]:
```
str
```

String can be combined in Python

[30]:
```
c = a + b + 'world!'
print(c)
```

```
hello world!
```

String variables can span multiple lines (this is because python recognizes the return character). To write a variablel spanning multiple lines you will need to act creative with the use of quotation marks ' '. You can use triple quotes `''' '''` to write a string that spans multiple lines:

[31]:
```
s = '''Hello World,
How are you?
Marvelous!'''
print(s)
```

```
Hello World,
How are you?
Marvelous!
```

1.3.7 6. NoneType

Python has another quirky variable type that is seldom used (or at least I seldom used it). The `NoneType`, this type stores a variable that does not have a specified type from the ones described above determined. A `NoneType` variable is defined by assign to it the value `None` (upper case), which is also a proprietary word in Python, just like `False` and `True`.

[32]:
```
a = None
type(a)
```

[32]:
```
NoneType
```

The value of a `NoneType` variable is `None`.

[33]:
```
print(a)
```

```
None
```

1.4 Basic math, strings and logic operators

1.4.1 Math Operators

Math operators implement basic forms of, ahem, operations with numbers. The operators allow to add numbers, multiply and divide them, suctract then

The table below (adapted from here) provides an overview of the foundamental Python math opertors.

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`
`**`	Exponentiation. It implements exponentials, power calculation on operators	`a**b` =10 to the power 20

Operator	Description	Example
// Floor Division	The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity)	9//2 = 4 and 9.0//2.0 = 4.0, -11//3 = -4, -11.0//3 = -4.0

Answer the following questions.

- What is the cube of 3.33? [Return you answers in the below]

[34]:
```
3.33**3
```

[34]:
```
36.926037
```

- Write code to perform the sum of `a = 3` and `b = 5.51` multiplied by `65.1233`? [Return you answers in the below]

[35]:
```
a = 3
b = 5.51
(a + b) * 65.1233
```

[35]:
```
554.199283
```

1.4.2 Logical Operators

Logical operators perform logical comparisons between variables and return a `Bool` valued output variables.

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

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

AD

Answer the following questions.

- Compare 3.33333 and 5.55, which one is bigger? [Return you answers in the cell below]

[36]:
```
3.33333 < 5.55
```

[36]:
```
True
```

- Is it true that 3.33333 and 5.55 are not equal? [Return you answers in the cell below]

[37]:
```
3.33333 != 5.55
```

[37]:
```
True
```

Additional reading This is an exhaustive tutorial covering more types of operators in python.