Introduction to Programming W2

Basics of data types and operators

2022/04/15

This class, we will learn about basic data types, variables, and operators. We will use the built-in `print()` function to display an input as text in the output, same way we did previous class.

1 Variables, numbers, and operators

Basic arithmetic operators:
Addition: `+`, Subtraction: `-`, Multiplication: `*`, Division: `/`, Exponentiation: `**`, Division and round: `//`, Modulo: `%`
In Python, the order of the operators holds. Try the following and observe the output.

```
print(3*2+6)
```

Variables are assigned using an equal sign, but mathematical equality and assignment are not the same. Type the following, run the code, and observe the output.

```
age = 32
half_age = age/2
print(age)
print(half_age)
```

Multiple variables can be assigned in one line. Try the following and observe the output.

```
a, b, c = 1, 2, 3
print(a*b*c)
```

The identifiers below are *keywords* of Python, and cannot be used as variable names:

```
False      await      else       import     pass
None       break      except     in         raise
True       class      finally    is         return
and        continue   for        lambda     try
as         def        from       nonlocal   while
assert     del        global     not        with
async      elif       if         or         yield
```

Based on the programming language, there can be different customs how to name variables. In Python, the most common way to name variables is to use lowercase letters with underscores to connect words, e.g., `class_size = 49`.

There are two built-in data types in Python used for storing numbers; integers and floating point values. Try the following code and observe the output.

```
a = int(3) #variable "a" is integer 3
print(a)
print(type(a))

a = float(4.2) #variable "a" is reassigned, and now is float 4.2
print(a)
print(type(a))

b = float(3) #this is actually 3.0
print(b)
print(type(b))
```

Above, the single hash sign `#` is used to *comment* our code inline. In Python, the built-in `type()` function can be used to check the type of any object.
The data type called *bool* or Boolean can hold one of the values `True` or `False`, encoded with numbers `1` and `0`. The following *comparison operators* can be used to get bool values:

Operator	Operation	Example	Bool
==	equal	3 == 4	False
!=	not equal	3 != 4	True
>	greater than	3 > 4	False
>=	greater than or equal	4 >= 3	True
<	less than	3 < 4	True
<=	less than or equal	4 <= 3	False

The following *logical operators* can be used to connect statements:

Operator	Operation	Example	Bool
`and`	True if both statements are True	3 < 4 `and` 5 < 4	False
`not`	evaluate and reverse the result	`not` 3 < 4	False
`or`	True if at least one of the statements is True	3 < 4 `or` 5 < 4	True

Try the following and observe the output.

```
a, b, c = 10, 15, 20
print(a < b)
print(not a < 15 and c != b)
```

2 Strings

The variable type *string* (type `str`) can be defined with double or single quotes. If the string includes quotation marks, you need to use the \ character to escape it, or use the one that is not used in the string. Try the following:

```
string_1 = 'holds a string'
string_2 = "this is also correct"
string_3 = 'Leibniz\'s law of continuity' #the \ character was used
string_4 = "Pascal's calculator" #different quotes are used
print(string_1)
print(string_2)
print(string_3)
print(string_4)
```

Indexing with square brackets can be used to access parts of the string. In Python, indexing starts at 0. Without removing the previous variable assignments, try the following:

```
print(string_1[0]) #string_1[0] is 'h' so it will print h
print(string_1[3]) #string_1[3] is 'd' so it will print d
```

Strings can be concatenated using the `+` operator. Try the following:

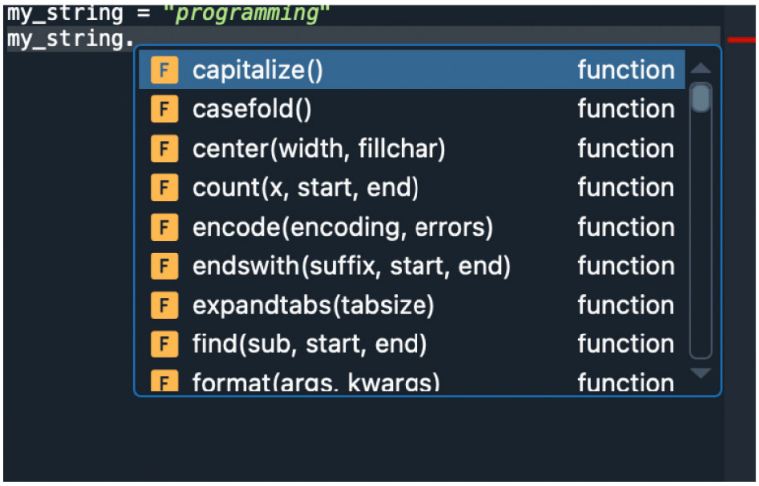
```
my_string_1 = "Introduction"
my_string_2 = "Programming"
print(my_string_1 + ' ' + 'to ' + my_string_2) #notice that we had to use whitespaces
```

The length of an object can be checked by the built-in `len()` function. In the case of a string, this will be the number of characters. Try the following:

```
print(len('Newton'))
print(len("12345")) #notice that despite using numbers, this is still a string
```

Slightly different than such functions, there are so-called *methods* in Python. Methods are specific to data type. Some require one or more *arguments*, same way as the `len()`, `print()`, or `type()` functions (inside the parentheses). Methods are accessed by dot notation. Using the Spyder IDE, one can check what methods a certain object/variable has. After typing the name of the variable, press dot and scroll through the methods (if the list disappears, you can

access it again by pressing Tab). You can select a method by double clicking on it from the list, or just simply typing it.



For example, the method `.capitalize()` will capitalize the first letter of the string, and it does not require arguments. Observe the output of the following code:

```
my_string = "programming"
print(my_string.capitalize())
```

On the other hand, the `.format()` method requires arguments. The brackets are replaced with the objects (not necessarily strings) passed into the method in the given order. Try the following code:

```
print('There are {} students in the class.'.format('49'))

no_students = 49 #notice that here no_students is an int
class_name = "Introduction to Programming"
print('There are {} students in the {} class.'.format(no_students, class_name))
```

3 File headers

When making a new Python file for the class (exercises, assignments), make sure that you include the following lines at the beginning of your code. The first line is only for macOS and Linux distributions (Python interpreter in the user path), the second is about the encoding. These two are automatically present when you make a new file in Spyder. We will learn more about what is a *docstring* (the text inside three double quotes) in later classes.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""
Author: student name
Student ID: student id number

Program description: The description of the program with your own words
"""
```

Exercise

Complete the following code based on the instructions in the comments. Students do not need to include the instructional comments, but encouraged to use their own words to explain the code in detail.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""
Author: student name
Student ID: student id number
```

```
"""Program description: Practicing variable assignment, type checking, operators, and string methods
"""
#assign three variables to numbers 10, 20, and 31, and use operators to get their average, and
#print the result
num_1 = 10
num_2 =
num_3 =
average =
print(average)
# print the the data type of variable average
print()
#using comparison operators, change the following statement into
#False at least 2 different ways
print(num_1 < num_2)

#assign the following string to a variable and print it: I don't know much about programming yet.
string_0 =
print()
#complete the following print function so the string reads: The 4th character is "o".
#bonus: use string indexing of string_0 to access the 4th character in the print function!
print()
#print the length of the following string
my_string = "apples"
print()
#use the format method after the dot to insert the words 'inventor' and 'mechanical engineer'
#into the string
print(Nikola Tesla was a Serbian-American {}, electrical engineer, {}.)
```