Chapter 2 - Data Types

Objectives:

- Learn about data types in Python including: integer, float, boolean and string
- Learn how to convert one data type into another
- learn how to use formatted printing

2.0 Every variable (and value) has a type

- **Integer** (int): represents positive or negative whole numbers like 3 or -512.
- **Floating point number** (float): represents real numbers like 3.14159 or -2.5.
- **Complex**: Python uses `j` as the complex unit, so `c = 2 + 3j` is a complex number (assuming you haven't redefined `j` as a variable).
- **Character string** (usually called “string”, str): text contained inside quotes like 'hello' or "howdy" (single and double quotes both work). The quote marks aren't printed when the string is displayed.
- **Boolean** (bool): variables that store True or False values. These variable are useful in loops and `if statements` that will be discussed later.

Let's define a few variables and examine their data types.

```
x = 4           # integer
y = 4.2         # float
c = 1 + 2j      # complex
a = 'dog'       # string
happy = True    # boolean
```

Notice that in Jupyter Notebooks, number values are colored dark green, strings are red and Boolean values are blue.

We can check the data type of a variable in two ways:

- Use the `type` function (hard way)
- Use the variable inspector (easy way)

Here's an example of how to display the data type of our 4 variables using the `print()` and `type()` functions (run it to see the results):

```
print('data type of x is',type(x))
print('data type of y is',type(y))
print('data type of c is',type(c))
print('data type of a is',type(a))
print('data type of happy is',type(happy))
```

The next section shows how to use the Variable Inspector.

2.1 The Variable Inspector

The **Variable Inspector** icon may be found on the bottom left corner of this window and it is labeled `{} Variables`. Click this icon to display the variable inspector. You should see a list of all the variables you have used in this tutorial, their values, and their

data types. While the Variable Inspector can be very helpful in debugging your code, be aware that it can slow down execution of complex code while it is open. Thus, you may want to only open it when it is needed.

The Variable Inspector can be very useful for debugging code, especially when you start writing complex programs.

2.2 Data types control what operations can be performed on a variable

The `+` operator has different effects on numbers and strings. Run the following code, for example:

```
x = 3
y = 4
print(x+y)     # the + operator adds number

a = 'Marie '   # notice we added a space at the end of 'Marie'
b = 'Curie'
print(a+b)     # the + operator concatenates strings
```

We see that the `+` operator:

- adds numbers (either floats or ints)
- concatenates strings. In other words, it sticks the two strings together to make a longer string

✅ Skill Check 1

See what effect the `*` operator has on strings

- Create a code cell
- assign a character string to a variable (for example, you could name it `my_string`)
- print your string variable as well as the value `10*my_string` .
- Your print statement should say something like 'If my_string = ___, then 10 * my_string = ___, where your code fills in the blanks ___.

2.3 Data Types can be converted into one another

2.3.1 Convert string to integer

A common situation when you might want to convert a string to an integer is when the `input()` statement is used to prompt the user for a number. Run the following:

```
result = input('Enter an integer:  ')                # input a number (but result is a string by default)
print('the data type of ',result,'is',type(result))  # print the data type of result
print('4 *',result,'=',4*result)                     # effect of multiplying result by 4
```

```
Enter an integer:  3
the data type of  3 is <class 'str'>
4 * 3 = 3333
```

Double-click (or enter) to edit

- The default data type returned by the `input` statement is a string.
- This can be confusing: even though the variable `result` shows its value to be `4` , we can't do any math with it yet because it is a string and not an integer or float.
- In the above code cell, we try multiplying our string variable `result` by 4 to get a surprising result!!

- To convert this string to an integer we can use `int()`:

```
result = input('Enter an integer:  ')         # input a number (data type is string)
n = int(result)                               # convert string to integer
print('the data type of ',n,'is',type(n))     # print data type of n
print('4 *',n,'=',4*n)                       # show that we can now use n in calculations
```

- We see that multiplication makes more sense now that we are multiplying an integer and not a string.

✅ Skill Check 2

Write some code to do the following:

- prompt the user for two numbers
- calculate the product of those numbers
- display the results with a message that has the form "2 x 3 = 6"
- hint: to enter two numbers, you'll need two input statements, one for each number.

As before, you'll need to create a code cell by clicking on "+Code" beneath this text cell.

🟩

2.3.2 Convert string to float

The `float()` function converts strings to floats:

```
a = '9.8'
g = float(a)
print('the data type of ',a,'is',type(a))
print('the data type of ',g,'is',type(g))
```

2.3.3 Convert integers and floats to strings

The simplest way to convert a number (either an integer or float) to a string is the `str()` function:

```
a = str(3.14)
print('the data type of ',a,'is',type(a))
```

The method described in the next section describes how to have more control over the appearance of string representations of numbers.

2.3.4 Convert float to integer

If we want to remove the digits to the right of the decimal point, we can just use the `int()` command like this:

```
p = int(2.999)
print(p)
```

However, if you want to round your float to the nearest integer, you can use `round()`:

```
r = round(2.999)
print(r)
```

Both of these methods convert floats to ints. You can verify this in the Variable Inspector or by using the `type()` command.

2.3.5 Convert Integer to Float

We can convert an iteger to a float by using the `float()` command:

```
q = float(3)
print("The data type of",q,"is",type(q))
```

✅ Skill Check 3

What happens when you add or multiply a float by an integer? Is the resulting data type float or int? Write some code to show an example of each. Create a code cell to try out your examples and create a text cell to explain your answer.

2.4 f-strings and formatted print

Let's suppose we are given the a number with more significant digits than we care about. We want to display our number keeping only two decimal places to the right of decimal place. We can do this with a formatted "f" string like this:

```
x = 316.227766       # float number showing 6 decimal places
x_str = f'{x:.2f}'   # string representation of x keeping 2 decimal places
print(x_str)         # display the truncated value
```

```
316.23
```

A formatted string has text inside a pair of quotes, but with the letter "f" out in front. We place a variable that we want to format inside curly braces followed by a formatting string. In the above example, the `:.2f` means we want to format the number as a float and keep 2 digits to the right of the decimal point.

If we just want to print the formatted number, we can just include the formatted string directly in the print function without the `x_str` variable:

```
print(f'{x:.2f}')    # simplified formatted print
```

```
316.23
```

Here are more examples for formatting floats:

```
print(f'1st example {x:.2f}')   # displays float with 2 decimal places
print(f'2nd example {x:.4f}')   # displays float with 4 decimal places
print(f'3rd example {x:10.2f}') # displays float with 5 decimal places and reserves total of 12 places
print(f'4th example {x:.2e}')   # scientific notation with 2 digits after the decimal
```

```
1st example 316.23
2nd example 316.2278
3rd example     316.23
4th example 3.16e+02
```

For integers you can specify the number of spaces set aside for columns of numbers. The formatting symbol for integers is "d". Heres's an example showing what happens when you print a set of numbers with different numbers of digits both with and without formatting. Since our biggest number has 5 digits, we'll reserve 5 digits for each data column using the format `{my_var:5d}`, where you would replace "`my_var`" with your variable name.

```
a = 2; b = 21043; c = 77; d =764
print(f"{a:5d}  {a:5d}      {a}  {a}")
print(f"{b:5d}  {b:5d}      {b}  {b}")
print(f"{c:5d}  {c:5d}      {c}  {c}")
print(f"{d:5d}  {d:5d}      {d}  {d}")
```

```
    2      2      2  2
21043  21043      21043  21043
   77     77      77  77
  764    764      764  764
```

Notice how the first two columns that are formated are nicely lined up (and right-justified), while the last two columns aren't. As we'll see later, formatted print statements are particularly useful when saving data to text files.

You can get fancy and include text and multiple formatted variables in the same print statement like this:

```
x = 4.334565   # x measurement
y = 12.4848    # y measurement
print(f'My measurements are x = {x:.2f} and y = {y:.2f}')
```

Remember, the variables go inside curly braces with the formatting.

✅ Skill Check 4

Write some code to prompt the user to enter a floating-point number. Break the number into its integer part (numbers to the left of the decimal) and the fractional part (numbers to the right of the decimal). Use a single print statement to display the original number, the whole part and the decimal part. Your print statement should produce a result that is easy to understand to the user.

✅ Skill Check 5

The following code block defines the x and y coordinates of an object. In the same code cell create a formatted print statement that uses these variables to display the position as a coordinate pair, where each position is displayed with 2 digits to the right of the decimal. Use the print statement to place the numbers inside parentheses and separated by a comma. Your final result should look like "(2.33, 5.95)". Hint: you can print the parentheses and comma by placing them in quotes, i.e. "(" and ",".

```
x = 2.33456    # x position in meters
y = 5.94843    # y position in meters
```

2.5 Possible errors

Unsupported operands

- If we try to do math with variables that are strings (or other non-numeric types), we can get the unsupported operand error:

```
a = "3"
b = "2"

c = a-b
```

- One possible solution is to convert the variable to an integer or float using the methods described in this chapter

Invalid literal

In this example, we try converting the string "cat" to an integer (which doesn't make sense). You can only convert string representations of numbers to integers or floats.

```
int("cat")
```

Key Points

- Every variable (and value) has a data type
- The Variable Inspector is useful for debugging code and quickly seeing the data type and values of variables
- Data types control what operations can be performed on a variable
- Data types can be converted into one another
- formatted f-strings allow fancy formatting of variables in a print statement

This tutorial is a modified adaptation of "Python for Physicists" © Software Carpentry