Day2_Data_and_Storage

July 23, 2021

Day 2: Introduction to Data Types and Storage

Today, we will be going over the fundamentals of coding. These fundamentals are often very similar across coding languages, so those of you who have coded before may recognize the patterns.

We will start at the beginning with variables and work our way up to strings. This will be very helpful as we move onto writing functions, loops, and working with data in subsequent lessons.

Goals for the day:

- Being able to do simple math or print in a cell
- Understand variables and data types
- Understand Booleans
- Begin using lists and indexing
- Understand strings and string manipulation
- Begin using dictionaries
- Go over built in python functions
- HAVE FUN

Functions Learned:

- Print to console: `print()`
- Length of string: `len()`
- Change variable to a Boolean (true/false) value: `bool(x)`
- Change to a complex number: `complex(x)`
- Change to a floating point: `float(x)`
- Change to an integer: `int(x)`
- Change to a string: `str(x)`

1.0 Simple math and printing in a cell

In our **coding cell** we can type out some operations and see the results in the line below. You can run a cell by pressing **Shift + Enter** or **Ctrl + Enter**. I use **Shift + Enter** because it will bring your cursor onto the next line of code to be run.

Some of the more complex math symbols are:

- `**` for exponentiation
- `//` for integer division

- % for modulo operator

[1]:
```
2 + 4/7
```

[1]:
```
2.571428571428571
```

[2]:
```
5*4 + 20
```

[2]:
```
40
```

[3]:
```
3**2
```

[3]:
```
9
```

We can print a sentence to the console:

[4]:
```
print("Hello World!")
```

```
Hello World!
```

2.0 Variables

This is fun, but doesn't get us very far unless you're trying to calculate a tip for your pizza delivery driver. When we want to work with data or start doing some calculations, we need to assign values to variables.

Variables can be described as 'boxes' you assign a value to, or a reference to a certain value. It's best practice to assign values to short and descriptive variables, and python has rules about naming.

- Can contain only numbers, letters, and underscores. Must begin with a letter or underscore
- Spaces are not allowed in variable names
- Variable names are case sensitive

[5]:
```
temp_c = 40
```

Notice how when we ran that line, nothing showed up. That's because it's being stored in the memory. We can `print` the variable using Python's built in function `print` to see the output:

[6]:
```
print(temp_c)
```

```
40
```

We can also change the variable to be something else, and it will overwrite the previous value:

[7]:
```
temp_c = 125
print(temp_c)
```

```
125
```

Python knows various types of data, the commonly used ones are:

- Integer numbers
- Floating point numbers

- Strings

We just saw integer numbers. This is an example of a floating point number:

[8]:
```
temp_c = 100.0
print(temp_c)
```

```
100.0
```

This is an example of a string:

[9]:
```
temp_c_text = 'temperature in celcius:'
print(temp_c_text)
```

```
temperature in celcius:
```

3.0 Commenting your code

Best practices when coding is leaving descriptive comments for yourself or collaborators. This becomes incredibly helpful when you have to come back to a script years later and you've forgotten completely what you've done.

Using the notebook style for code can be helpful, but when you are building more complicated scripts it can get too messy. So we can comment code by putting a # symbol and Python will ignore anything after that

[10]:
```
temp_c = 15.5 # temperature in celcius
print(temp_c) # printing the temperature in celcius

# you can also take notes in your own code during this lesson

# katie says that it's best practice
```

```
15.5
```

4.0 Manipulating variables in Python

We can print multiple variables together:

[11]:
```
print(temp_c_text, temp_c)
```

```
temperature in celcius: 15.5
```

We can do arithmetic on the variable and assign it to a new variable:

[12]:
```
temp_f = temp_c * (9/5) + 32 # converting temperature in celcius to fahrenheit
print(temp_f)
```

```
59.900000000000006
```

You can assign multiple variables using multiple assignment:

[13]:
```
temp1, temp2, temp3 = 10,20,35 # setting the values for three different␣
 ↪temperatures
print(temp3,temp2,temp1)
```

```
35 20 10
```

4.1 Coding is weird

So when coding, you can do a weird thing that doesn't work in math, where you can update a variable using that variable. For example, if I am trying to update a variable `x` to be `x+1`, I can write it as follows:

Note: I can do this all in one coding cell. So far we have only seen very short lines of code in one cell, but you can write big blocks of code and run it all together.

I'm also using print() here multiple times so that I can show what the value of `x` as it gets updated in the cell

[14]:
```
x = 5
print(x)

x = x + 1
print(x)

x = x + 1
print(x)
```

```
5
6
7
```

5.0 Coding check in

Imagine you're at the gym and you are doing your bench press with weights in kilograms, but you want to write down your sets in lbs. Set three variables of weights in kg (ex. 5, 10, 25) and then convert it to weight in lbs (1 kg = 2.2 lbs).

Note: you will convert them all separately.

Tip: If you are stuck, try with one weight first

BONUS: Comment your code throughout

[15]:
```
# empty cell for you to type in the coding check in

# 1. set three weight in kg


# 2. convert the weight in kg to lbs


```

```
# 3. print out the weight in lbs
```

5.1 Answer (I'm going to do it for you, so don't look)!:

[16]:
```
## NOTE: there are several ways you could have accomplished this
## everyones code always looks different, this is normal

weight1, weight2, weight3 = 5,10,25 # setting the value for three weights

conversion = 2.2 # I'm using a variable as a conversion to make sure I don't do␣
 ↪a typo later

weight_lbs_1 = weight1*conversion # converting the weights from kg to lbs
weight_lbs_2 = weight2*conversion
weight_lbs_3 = weight3*conversion

print(weight_lbs_1, weight_lbs_2, weight_lbs_3) # printing out the weights
```

```
11.0 22.0 55.00000000000001
```

6.0 Boolean variables

These are binary variables, which can be either true or false. We can use this as a tool to interrogate Python about our variables, and later we can use them in more complex ways.

There are several different tests we can do. These are including:

- == if something is equal to
- > greater than
- >= greater than or equal to
- < less than
- <= less than or equal to
- != not equal to

First, what was our temp1 value from before:

[17]:
```
temp1 = 10

print(temp1)
```

```
10
```

Now we can ask if temp1 is greater than 0, to see what type of jacket we need to go outside in:

[18]:
```
temp1 > 0
```

[18]:
```
True
```

We can ask if a certain value is **exactly equivalent** to some value using two equal signs. Notice this is different than our normal assignment operator, as it's like asking the question "is this equal to that":

[19]:
```
temp1 == 20
```

[19]:
```
False
```

Let's try with another temperature variable, `temp5`, to see if this is above zero:

[20]:
```
temp5 > 0 # we are expecting an error
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-20-0a43ce23ecf5> in <module>
----> 1 temp5 > 0 # we are expecting an error

NameError: name 'temp5' is not defined
```

When we tried to interrogate `temp5`, notice we get the error: 'temp5' is not defined. That's because I've never defined temp5 to be anything. This is to give you an idea of the kinds of errors Python will give you.

To see all the variables (and functions) you've created during this session, you can use the built in function `who`:

[21]:
```
who
```

```
conversion       temp1   temp2   temp3   temp_c  temp_c_text     temp_f  weight1
weight2
weight3  weight_lbs_1    weight_lbs_2    weight_lbs_3    x
```

7.0 Strings

There are a lot of built in ways to manipulate strings, which will come in very handy when working with data. We can get the length of the string, we can manipulate it, etc.

[22]:
```
my_string = "This is my string"
print(my_string)
```

```
This is my string
```

The first thing we can do is to get the length of the string using the built in function `len`:

[23]:
```
len(my_string) # prints the length
```

[23]:
```
17
```

Next, we can append a string onto another one by using a +:

[24]:
```
my_string + " and it is long"
```

[24]:
```
'This is my string and it is long'
```

We can do more complex string manipulation using the built in functions. In python, you can use a function by including a period and then the function `replace` and parentheses afterwards:

[25]:
```
my_string.replace("string", "sentence") # replace one word with another
```

[25]:
```
'This is my sentence'
```

That last command printed out the new string, but now we want to permanently change it to be the new one, so we update our variable `my_string` using `replace`

[26]:
```
my_string = my_string.replace("string","sentence")
print(my_string)
```

```
This is my sentence
```

[27]:
```
my_string.split() # split a longer string into several shorter ones
```

[27]:
```
['This', 'is', 'my', 'sentence']
```

[28]:
```
print(my_string.upper()) # replace with all uppercase
print(my_string.lower()) # replace with all lowercase
```

```
THIS IS MY SENTENCE
this is my sentence
```

[29]:
```
my_string.title() # capitalizes the first letter of each word
```

[29]:
```
'This Is My Sentence'
```

8.0 Coding check in

Create two strings with your name set up as "My name is [first name]" and "[last name]". Create a new string with them together together, then replace the last name to the celebrity crush of your choice, then print them out all caps.

I will be using Zac Efron for my celebrity crush (and I'm not afraid to admit it)

[30]:
```
## where you can put your coding check in code
```

8.1 Answer (I'm going to do it for you)!

[31]:
```
first_name = "My name is Katie "
last_name = "Dixon"
my_name = first_name + last_name
print(my_name)
my_name = my_name.replace("Dixon", "Efron")
print(my_name)
my_name.upper()
```

```
My name is Katie Dixon
My name is Katie Efron
```

[31]:
```
'MY NAME IS KATIE EFRON'
```

9.0 Lists and indexing

9.1 Lists are ordered collections of values, which are separated by a comma. You can make lists of numbers, letters of the alphabet, booleans, or strings. The strings can include multiple different data types as well. First, let's make a list of animals:

[32]:
```
animals = ['cat', 'dog', 'squid', 'moose', 'falcon']
print(animals)
```

```
['cat', 'dog', 'squid', 'moose', 'falcon']
```

We can get the elements of the list through indexing, which starts at 0 and is limited by the length of the string. This is different from some other langages (e.g. R) which starts at 1.

[33]:
```
animals[0]
```

[33]:
```
'cat'
```

[34]:
```
animals[6] # there will be an error
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-34-16c23d9102a0> in <module>
----> 1 animals[6] # there will be an error

IndexError: list index out of range
```

We can also select certain elements of a list by using a colon :. The syntax is start:stop and does not include the stop value. So, if we do 1:5, we get index 1,2,3, and 4.

[35]:
```
animals[2:5] # slicing the index 2 through 5 to get 2,3,4
```

[35]:
```
['squid', 'moose', 'falcon']
```

We can also do it so that we have either `start:` or `:end` where we don't specify the other value. * `start:` all elements including and after this starting point * `:end` all elements before this ending point

[36]:
```
animals[2:] # index 2 and on
```

[36]:
```
['squid', 'moose', 'falcon']
```

[37]:
```
animals[:2] # everything before index 2
```

[37]:
```
['cat', 'dog']
```

We can get the last element of the list by using -1:

[38]:
```
animals[-1]
```

[38]:
```
'falcon'
```

List are also 'mutable', which means we can change elements of the list without impacting the others:

[39]:
```
print(animals) # printing to remember what they are
animals[1] = 'wolf' # changing the first index of the list
print(animals)
```

```
['cat', 'dog', 'squid', 'moose', 'falcon']
['cat', 'wolf', 'squid', 'moose', 'falcon']
```

9.2 Adding lists together

We can create a second list (of more animals) and then combine them using a +

[40]:
```
animals2 = ['spider', 'orca', 'squid']
my_favorite = animals + animals2
print(my_favorite)
```

```
['cat', 'wolf', 'squid', 'moose', 'falcon', 'spider', 'orca', 'squid']
```

We can add elements to the list using `append`

[41]:
```
my_favorite.append('llama')
print(my_favorite)
```

```
['cat', 'wolf', 'squid', 'moose', 'falcon', 'spider', 'orca', 'squid', 'llama']
```

We can get the index (aka position) of an element in the string using `index`:

[42]:
```
my_favorite.index('moose') # getting the index of the value
```

[42]:
```
3
```

We can count the number of times a number or string is in the list using `count`

[43]:
```
my_favorite.count('squid') # the number of occurances of the value in parentheses
```

[43]:
```
2
```

We can reverse the string using `reverse`

[44]:
```
my_favorite.reverse() # this permanently reverses it
print(my_favorite)
```

```
['llama', 'squid', 'orca', 'spider', 'falcon', 'moose', 'squid', 'wolf', 'cat']
```

We can also do some subsetting at a small scale, and Amanda and Maria will give you better ways to do this later on! We can take different elements and make them into a new list

[45]:
```
ocean_animals = my_favorite[1], my_favorite[2]
print(ocean_animals)
```

```
('squid', 'orca')
```

9.3 Coding check in

- Make a list of 5 vegetables.
- Add a 6th vegetable to the list
- Change the fifth vegetable in the list to be something else
- Call up the first and last element and make a new variable with these in it.
  - This new variable is called a `tuple` and is not allowed to be changed. Google list vs. tuple if you want more information

[46]:
```
## coding check in here!
```

9.4 Answer to coding check in

[47]:
```
veggies = ['broccoli', 'asparagus', 'peas', 'green bean', 'carrot'] # my veggie␣
 ↪string
print(veggies)

veggies.append('corn') # adding corn to my list
print(veggies)

veggies[4] = 'eggplant' # changing green bean to eggplant, remember indexing␣
 ↪starts at zero
print(veggies)

favorite_veggies = veggies[0],veggies[-1] # first and last elements
print(favorite_veggies)
```

```
['broccoli', 'asparagus', 'peas', 'green bean', 'carrot']
['broccoli', 'asparagus', 'peas', 'green bean', 'carrot', 'corn']
['broccoli', 'asparagus', 'peas', 'green bean', 'eggplant', 'corn']
('broccoli', 'corn')
```

10.0 Dictionaries!

Dictionaries are unique data structures, which maps a key to a related values. So you can look for a key and get a value, this is useful for functions, dataframe manipulation, and so much more. We might want to link together several values, and we can use the function `dict` and `zip` to combine them together:

[48]:
```
# list of keys
my_keys=['a','b','c']

# list of values
```

```
my_values=[1,2,3]

#join the list using zip
alpha_num_dict=dict(zip(my_keys,my_values))

print(alpha_num_dict)
```

```
{'a': 1, 'b': 2, 'c': 3}
```

So now we have each key mapped to a value, so a is mapped to 1, etc. We can also do this in fewer lines (if we don't have that much to write):

[49]:
```
### Dictionaries relate keys to values, you look for a key and get a value
## manually enter the keys and values to dict directly
alpha_num_dict2=dict([('a',1),('b',2),('c',3)])
print(alpha_num_dict2)
```

```
{'a': 1, 'b': 2, 'c': 3}
```

This is useful because we can then use the key to call up a value using the function `get`:

[50]:
```
alpha_num_dict2.get('b')
```

[50]:
```
2
```

11.0 Coding check in

Write a dictionary linking together your four closest friends and their favorite animals. You can do so by making a list of friends and a list of animals and then zipping them together. Then call up one of their favorite animals using their name!

[51]:
```
## here is where you can write your code for the coding check in
```

11.1 Answer (don't check until you've tried)

[52]:
```
# creating a list of people and a list of animals
friends = ["Katie", "Alex", "Maria", "Amanda"]
animals = ["Cheetah", "Falcon", "Penguin", "Dolphin"]

# zipping the two lists together to create a dictionary
animal_dictionary = dict(zip(friends,animals))

# get Maria's favorite animal
animal_dictionary.get('Maria')
```

[52]:
```
'Penguin'
```

12.0 Built in functions

Some functions are already built into python, others you will have to load using packages.

You may have noticed that we are using the built in function `print` a lot. There is a reason for this that will become more important later. When using markdown/ notebook formats, the cell will only return on thing, so if you use several functions that return values, like `min` or `max`, it will only return the most recent one. By doing `print` it will print multiple things to the command line, to illustrate the answer.

[53]:
```
y = -15.95

abs(y) # absolute value
round(y,0)

# only returns the last thing we did, which was rounding
```

[53]:
```
-16.0
```

[54]:
```
y = -15.95

print(abs(y))
print(round(y,0))

# now we will see both answers
```

```
15.95
-16.0
```

We can also do things to lists of values:

[55]:
```
# for a list of values
x = [5,4,10,3,100]
print(max(x)) # largest value (input string, list, set, tuple, ...)
print(min(x)) # smallest value
print(sum(x)) # sum all values
print(len(x)) # length of list
```

```
100
3
122
5
```

Getting the length of a list is very important, and we will touch on it later when doing `for` loops! We can also perform examples of casting, which means to change the data type from one to another. We start with our value 5.5.

Examples of casting, which means changing the data type from one to another:

[56]:
```
x = 5.5

print(bool(x)) # to Boolean
print(complex(x)) # to complex number
print(float(x)) # to floating point
```

```
print(int(x)) # to integer
print(str(x)) # to string
```

```
True
(5.5+0j)
5.5
5
5.5
```

The Boolean value means almost nothing and you can't really tell the last one is a string, but this can be very useful in your research, especially if your data gets read in as the wrong format (this definitely happens!)

Here is how to check the structure of a variable:

[57]:
```
# the first is a float value (a number)
print(type(x))

# we will now make it a string using str
y = str(x)
print(type(y))

print(y)
```

```
<class 'float'>
<class 'str'>
5.5
```

Most of these only work on numeric variables, but Python can also cast strings that include numbers to numeric values:

[58]:
```
x = "5.5"
print(type(x)) # checking the type of x

x = float(x)
print(type(x)) # checking the type of x after we hopefully converted it
```

```
<class 'str'>
<class 'float'>
```

13 Coding check in

I'll provide you with a list of heights. Find the max and the min for the list of heights. Then, see if you can find the mean value. There isn't a built in function for mean, but you have all you need with `sum` and `len`.

Mutate the 3th index to be someone very tall (85 inches?). How does that change the mean?

[59]:
```
### space to write check in

heights_inches = [59, 80, 64, 55, 74, 70, 45, 66, 54, 64, 60, 61, 70]
```

13.1 Check in answer

[60]:
```
heights_inches = [59, 80, 64, 55, 74, 70, 45, 66, 54, 64, 60, 61, 70]

print(max(heights_inches))
print(min(heights_inches))

print(sum(heights_inches)/len(heights_inches))

heights_inches[3] = 85

print(sum(heights_inches)/len(heights_inches))
```

```
80
45
63.23076923076923
65.53846153846153
```