Week 2: Python Data Types

DSUA111: Data Science for Everyone, NYU, Fall 2020

TA Jeff, jpj251@nyu.edu

https://github.com/jpowerj/dsua111-sections
(https://github.com/jpowerj/dsua111-sections)

Data Types

1. ...What's a "data type"?
2. Integers (`int`)
3. Floats (`float`)
4. Strings (`str`)
5. Boolean Values (`bool`)
6. Functions
7. Lists (`list`)
8. NumPy Data Types

1. Integers (`int`)

Integers are exactly what they sound like, they are whole numbers, - $-\infty \ldots -3, -1, 0, 1, 2, 3 \ldots \infty$

Only whole numbers can be represented in Python as integers

In [6]:
```
foo = -7
```

In [7]:
```
print(foo)
```

```
-7
```

In [8]:
```
foo
```

Out[8]:
```
-7
```

In [9]:
```
new_foo = 10
```

In [10]:
```
new_foo
```

Out[10]:
```
10
```

All the arithmetic we did before works on ints!

2. Floats (`float`)

Floats are numbers with decimals

All (rational) real numbers can be represented as floats

In [11]:
```
bar = 5.5
```

In [12]:
```
bar
```

Out[12]:
```
5.5
```

In [13]:
```
check_var = 5.0
```

In [14]:
```
check_var
```

Out[14]:
```
5.0
```

In [15]:
```
bar_alt = 5
```

Importantly, floats and integers plays nice together

In [16]:
```
bar_alt + bar
```

Out[16]:
```
10.5
```

But notice, an integer plus a float makes a float. Generally speaking, python will work when two data types would make sense if they worked together.

But the output will usually be the more complex data type.

All the arithmetic we did before works on floats!

3. Strings (`str`)

Strings are how we represent text

Just put them in quotes (double or single!)

In [17]:
```
string_var = 'This is my string'
```

In [18]:
```
print(string_var)
```

```
This is my string
```

In [19]:
```
string_var
```

Out[19]:
```
'This is my string'
```

In [20]:
```
print("hello world")
```

```
hello world
```

Ints and floats can also be represented as strings

In [21]:
```
bar_string = '-7'
foo_string = '5.0'
```

In [22]:
```
bar_string
```

Out[22]:
```
'-7'
```

In [23]:
```
foo_string
```

Out[23]:
```
'5.0'
```

In [24]:
```
bar_string + foo_string
```

Out[24]:
```
'-75.0'
```

Can strings and floats or ints play nice?

In [25]:
```
bar_string*5
```

Out[25]:
```
'-7-7-7-7-7'
```

In [26]:
```
bar
```

Out[26]:
```
5.5
```

In [27]:
```
bar_string*bar
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-27-d46a3ac220f1> in <module>
----> 1 bar_string*bar

TypeError: can't multiply sequence by non-int of type 'float'
```

When we multiply a string by an int, it just repeats the string that many times

But floats don't work. How do you repeat "hello" 3.76 times?

In [28]:
```
foo
```

Out[28]:
```
-7
```

In [29]:
```
foo_string
```

Out[29]:
```
'5.0'
```

In [30]:
```
foo + foo_string
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-30-9857000bcc8c> in <module>
----> 1 foo + foo_string

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

This also doesn't work.

But this does!

In [31]:
```
foo_string
```

Out[31]:
```
'5.0'
```

In [32]:
```
bar_string
```

Out[32]:
```
'-7'
```

In [33]:
```
foo_string + bar_string
```

Out[33]:
```
'5.0-7'
```

It just adds them together!

There are other simply ways to interact with string, but this is the most important. Feel free to explore and experiment more.

4. Boolean Values (bool)

Bools are values of either True or False

variables can we assigned to bools OR python will return bools when we do what is called an evaluations statement

In [34]:
```
bool_var = False
```

In [35]:
```
bool_var
```

Out[35]:
```
False
```

In [36]:
```
var1 = 10
var2 = 20
```

In [37]:
```
var1 == var2
```

Out[37]:
```
False
```

4. Boolean Values (bool) Continued

In [38]:
```
var1 != var2
```

Out[38]:
```
True
```

In [39]:
```
var1 < var2
```

Out[39]:
```
True
```

In [40]:
```
var1 <= var2
```

Out[40]:
```
True
```

In [41]:
```
var1 >= var2
```

Out[41]:
```
False
```

Importantly, Bools are NOT strings.

A bool is True or False (with capitals), not "true" or "True" or "False" or "false"

In [42]:
```
new_var = "True"
```

In [43]:
```
new_var
```

Out[43]:
```
'True'
```

The above is NOT a bool - it is a string!

Notice, we can do evaulation with things that aren't numbers

In [44]:
```
my_name = "Jeff"
```

In [45]:
```
my_other_name = "Jeff"
```

In [46]:
```
my_name == my_other_name
```

Out[46]: True

In [47]:
```
my_full_name = "Jeffrey"
```

In [48]:
```
my_name == my_full_name
```

Out[48]: False

In [49]:
```
my_name in my_full_name
```

Out[49]: True

Simple Functions

So how do you know what type of data a variable is without looking at it?

In [50]:
```
foo
```

Out[50]:
```
-7
```

foo looooooooks like an int, but how do we know?

We can use a simply built in function, but what is a function?

5. Functions (`function`)

Functions run code, and you can pass (give) objects to them, and they will output things.

Let's see an example, the type function.

It looks like this: `type()`

here is how you use it:

The `type()` function

In [51]:
```
type(foo)
```

Out[51]:
```
int
```

Notice that type is green (that means python has reserved that word, you can't assign type as a variable, mean you can't do something like type = 5).

I passed (or handed, or gave) the variable foo to type, and then type figures out what data type it is, and then it outputs that

In [49]:
```
foo_string
```

Out[49]:
```
'5.0'
```

In [50]:
```
type(foo_string)
```

Out[50]:
```
str
```

type is also built in! That means python automatically has this function.

In the future, you will write functions, and import (basically borrow) the functions other people have written. Those functions will not be built in, because they don't automatically come with python for everyone.

importantly, you can assign the the things that functions output to variables, for example:

In [51]:
```
foo_type = type(foo)
```

In [52]:
```
foo_type
```

Out[52]:
```
int
```

What kind of type is foo_type?

In [53]:
```
type(foo_type)
```

Out[53]:
```
type
```

So what if we want to convert data types, like a string to a float (or vice versa)?

Simple, built in converters!

1. str() <- converts input to a string
2. int() <- converts input to an int
3. float() <- converts input to a float

In [54]:
```
foo_string
```

Out[54]:
```
'5.0'
```

In [55]:
```
float(foo_string)
```

Out[55]:
```
5.0
```

In [56]:
```
int(foo_string)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-56-979cc2db2ab9> in <module>
----> 1 int(foo_string)

ValueError: invalid literal for int() with base 10: '5.0'
```

In [57]:
```
int("1254")
```

Out[57]:
```
1254
```

In [58]:
```
my_int_str = '5'
int(my_int_str)
```

Out[58]:
```
5
```

In [59]:
```
int(5.0)
```

Out[59]:
```
5
```

It's a bit too much for python to go from '5.0' to 5, but we can nest fuctions!

Just put one inside the other, and python starts on the inside and goes out (PEMDAS)

In [60]:
```
foo_string
```

Out[60]:
```
'5.0'
```

In [61]:
```
float(foo_string)
```

Out[61]:
```
5.0
```

In [62]:
```
int(float(foo_string))
```

Out[62]:
```
5
```

In [63]:
```
float(foo_string)
```

Out[63]:
```
5.0
```

In [64]:
```
int(float(foo_string))
```

Out[64]:
```
5
```

Notice, this won't work if it doesn't make sense:

In [65]:
```
new_string = "not a number"
```

In [66]:
```
float(new_string)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-66-6de1d62688ec> in <module>
----> 1 float(new_string)

ValueError: could not convert string to float: 'not a number'
```

And sometimes, python has some default behavior.

When int() gets a float, it just takes everying to the left of the decimal (i.e., it always rounds down)

In [67]:
```
new_float = 5.999999999
```

In [68]:
```
int(new_float)
```

Out[68]:
```
5
```

6. Lists (`list`)

Lists are a datatype that can hold, store or otherwise be a place to put other data!

Let's create a list

In [69]:
```
my_list = []
```

In [70]:
```
my_list
```

Out[70]:
```
[]
```

my_list is an empty list.

you create lists with SQUARE brackets, i.e., [and]

Things are seperated in lists by commas

In [71]:
```
my_list2 = [1,23]
```

In [72]:
```
my_list2
```

Out[72]:
```
[1, 23]
```

my_list2 is a list, with three items, all of which are integers.

Lists can store ANY object, and they don't all have to be the same type

In [73]:
```
mix_list = [8,5.0,'5.0']
```

In [74]:
```
mix_list
```

Out[74]:
```
[8, 5.0, '5.0']
```

What if we want to get one item out of a list?

For example, what if we wanted to get the float 5.0 from mix_list?

Lists are ordered. Python will always remember the order of the elements in a list (this is not true of some other data types)

So all we have to do to get a specific item from a list, is tell python which number item we want.

How do we do that?

For lists, we just tell python the index of the item in list, i.e., whatever number it is.

BUT, python doesn't count 1,2,3,4.

Python indexs from 0. Python counts 0,1,2,3,4.

That means 5.0 is indexed to 1 (not 2), while 8 is indexed to 0

Remeber, indexing starts at 0 (NOT 1)

So how do we tell python, give us element 1 (the SECOND element) from mix_list? (or any list)

```
In [75]: mix_list
Out[75]: [8, 5.0, '5.0']

In [76]: mix_list[1]
Out[76]: 5.0

In [77]: mix_list[0]
Out[77]: 8

In [78]: mix_list[2]
Out[78]: '5.0'

In [79]: mix_list[5]
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-79-efefc08ff73a> in <module>
----> 1 mix_list[5]

IndexError: list index out of range
```

In [80]:
```
mix_list[-2]
```

Out[80]:
```
5.0
```

Ok, so now we can get at the items in lists, and we can assign those values to new variables

In [81]:
```
elem_2 = mix_list[2]
```

In [82]:
```
elem_2
```

Out[82]:
```
'5.0'
```

Can we change elements in the list? Yes, yes we can!

Lists are mutable (changable), but note that not all datatpyes are!

In [83]:
```
mix_list[2] ="A different string"
```

In [84]:
```
mix_list
```

Out[84]:
```
[8, 5.0, 'A different string']
```

How do we tell how long is? i.e., how many elements are in a list?

Use len(), len stands for length!

In [85]:
```
len(mix_list)
```

Out[85]:
```
3
```

Finally, there are some built in functions that work on lists that have only numbers (integers and floats)

They are max(), min() and mean() and they do what they say they do!

In [86]:
```
number_list = [423,34,1]
```

In [87]:
```
sum(number_list)
```

Out[87]: 458

In [88]:
```
min(number_list)
```

Out[88]: 1

In [89]:
```
max(number_list)
```

Out[89]: 423

How do we find the mean or average?

In [90]:
```
sum(number_list)/len(number_list)
```

Out[90]:
```
152.66666666666666
```

Removing things from list

In [91]:
```
number_list = [423,34,1, 423]
```

In [92]:
```
number_list
```

Out[92]:
```
[423, 34, 1, 423]
```

In [93]:
```
number_list.remove(423)
```

In [94]:
```
number_list
```

Out[94]:
```
[34, 1, 423]
```

In [95]:
```
del number_list[1]
```

In [96]:
```
number_list
```

Out[96]:
```
[34, 423]
```

In [97]:
```
number_list
```

Out[97]:
```
[34, 423]
```

In [98]:
```
number_list.append(27)
```

In [99]:
```
number_list
```

Out[99]:
```
[34, 423, 27]
```

7. NumPy Data Types

"Numerical Python".

Its an outside package that gives us lots and lots of convenient ways to work with data

If you've ever worked with vectors, matrices, or linear algebra, numpy is basically how we work with those things

If you don't know those things, no worries!

Before we can use numpy we have to import it

In [100]:
```
import numpy as np
```

Basically, we import numpy, but we ask python to abbreviate it to np (for convience)

Whenever we want to use some functions from numpy, we just start with np

For now, all we want to use are numpy arrays

Arrays are alot like lists, but they are designed to work with numbers

Here is how we create one

In [101]:
```
my_arr = np.array([1,2,3])
```

In [102]:
```
my_arr
```

Out[102]:
```
array([1, 2, 3])
```

We call numpy, call the array function, and then pass it a list of numbers

And we can do cool stuff with it

In [103]:
```
my_arr*5
```

Out[103]:
```
array([ 5, 10, 15])
```

In [104]:
```
my_arr**3
```

Out[104]:
```
array([ 1,  8, 27], dtype=int32)
```

This is very different than if we just did this to a normal list

In [105]:
```
num_list = [1,2,3]
```

In [106]:
```
num_list*5
```

Out[106]:
```
[1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]
```

In [107]:
```
my_list**5
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-107-60150ab04e48> in <module>
----> 1 my_list**5

TypeError: unsupported operand type(s) for ** or pow(): 'list' and 'int'
```

We even get more functionality

numpy has a mean function (np.mean())

In [108]:
```
np.mean(my_arr)
```

Out[108]:
```
2.0
```