

# Week 2: Python Data Types

DSUA111: Data Science for Everyone, NYU, Fall 2020

TA Jeff, [jpj251@nyu.edu](mailto:jpj251@nyu.edu)

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

# Data Types

...what's a "data type"?

# 1. Integers

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

Only whole numbers can be represented in Python as integers

```
In [1]: foo = -7
```

```
In [2]: print(foo)
```

```
-7
```

```
In [4]: foo
```

```
Out[4]: -7
```

```
In [5]: new_foo = 10
```

```
In [6]: new_foo
```

```
Out[6]: 10
```

All the arithmetic we did before works on ints!

## 2. Floats

Floats are numbers with decimals

All (rational) real numbers can be represented as floats

```
In [7]: bar = 5.5
```

```
In [8]: bar
```

```
Out[8]: 5.5
```

```
In [9]: check_var = 5.0
```

```
In [10]: check_var
```

```
Out[10]: 5.0
```

```
In [11]: bar_alt = 5
```

## Importantly, floats and integers plays nice together

```
In [12]: bar_alt + bar
```

```
Out[12]: 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

Strings are how we represent text

Just put them in quotes (double or single!)

```
In [13]: string_var = 'This is my string'
```

```
In [14]: print(string_var)
```

```
This is my string
```

```
In [15]: string_var
```

```
Out[15]: 'This is my string'
```

```
In [17]: print("hello world")
```

```
File "<ipython-input-17-278f78f45249>", line 1
```

```
    print("hello world")
```

```
        ^
```

```
SyntaxError: EOL while scanning string literal
```

# Ints and floats can also be represented as strings

```
In [18]: bar_string = '-7'  
         foo_string = '5.0'
```

```
In [19]: bar_string
```

```
Out[19]: '-7'
```

```
In [20]: foo_string
```

```
Out[20]: '5.0'
```

```
In [21]: bar_string + foo_string
```

```
Out[21]: '-75.0'
```

# Can strings and floats or ints play nice?

```
In [22]: bar_string*5
```

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

```
In [23]: bar
```

```
Out[23]: 5.5
```

```
In [24]: bar_string*bar
```

```
-----  
TypeError                                Traceback (most recent call last)  
<ipython-input-24-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 [25]: foo
```

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

```
In [26]: foo_string
```

```
Out[26]: '5.0'
```

```
In [27]: foo + foo_string
```

```
-----  
TypeError                                Traceback (most recent call last)  
<ipython-input-27-9857000bcc8c> in <module>  
----> 1 foo + foo_string  
  
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

This also doesn't work.

But this does!



```
In [28]: foo_string
```

```
Out[28]: '5.0'
```

```
In [29]: bar_string
```

```
Out[29]: '-7'
```

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

```
Out[30]: '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. Bools

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 [33]: bool_var = False
```

```
In [32]: bool_var
```

```
Out[32]: True
```

```
In [34]: var1 = 10  
var2 = 20
```

```
In [35]: var1 == var2
```

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

```
In [36]: var1 != var2
```

```
Out[36]: True
```

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

```
Out[37]: True
```

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

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

```
In [39]: var1 >= var2
```

```
Out[39]: False
```

Importantly, Bools are NOT strings.

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

```
In [40]: new_var = "True"
```

```
In [41]: new_var
```

```
Out[41]: 'True'
```

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

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

```
In [42]: my_name = "Will"
```

```
In [43]: my_other_name = "Will"
```

```
In [44]: my_name == my_other_name
```

```
Out[44]: True
```

```
In [45]: my_full_name = "William"
```

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

```
Out[46]: False
```

```
In [47]: my_name in my_full_name
```

```
Out[47]: True
```

## Simple Functions

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

```
In [48]: foo
```

```
Out[48]: -7
```

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

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

```
In [49]: type(foo)
```

```
Out[49]: 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 [50]: foo_string
```

```
Out[50]: '5.0'
```

```
In [52]: type(foo_string)
```

```
Out[52]: 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,

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

```
In [55]: foo_type
```

```
Out[55]: int
```

What kind of type is foo\_type?

```
In [56]: type(foo_type)
```

```
Out[56]: 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 [59]: foo_string
```

```
Out[59]: '5.0'
```

```
In [60]: float(foo_string)
```

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

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

```
-----  
ValueError                                Traceback (most recent call last)  
<ipython-input-61-979cc2db2ab9> in <module>  
----> 1 int(foo_string)  
  
ValueError: invalid literal for int() with base 10: '5.0'
```

```
In [64]: int("1254")
```

```
Out[64]: 1254
```

```
In [65]: my_int_str = '5'  
int(my_int_str)
```

```
Out[65]: 5
```

```
In [66]: int(5.0)
```

```
Out[66]: 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 [67]: foo_string
```

```
Out[67]: '5.0'
```

```
In [68]: float(foo_string)
```

```
Out[68]: 5.0
```

```
In [69]: int(float(foo_string))
```

```
Out[69]: 5
```

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

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

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

```
In [70]: new_string = "not a number"
```

```
In [71]: float(new_string)
```

```
-----  
ValueError                                Traceback (most recent call last)  
<ipython-input-71-6de1d62688ec> in <module>  
----> 1 float(new_string)  
  
ValueError: could not convert string to float: 'not a number'
```

**And sometimes, python has some default behavior.**

```
In [72]: new_float = 5.999999999
```

```
In [73]: int(new_float)
```

```
Out[73]: 5
```

# Lists!

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



Let's create a list

```
In [81]: my_list = []
```

```
In [84]: my_list
```

```
Out[84]: []
```

**my\_list** is an empty list.

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

Things are separated in lists by commas

```
In [85]: my_list2 = [1,23]
```

```
In [86]: my_list2
```

```
Out[86]: [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 [87]: mix_list = [8,5.0,'5.0']
```

```
In [88]: mix_list
```

```
Out[88]: [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 indexes 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



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

```
In [89]: mix_list
```

```
Out[89]: [8, 5.0, '5.0']
```

```
In [90]: mix_list[1]
```

```
Out[90]: 5.0
```

```
In [91]: mix_list[0]
```

```
Out[91]: 8
```

```
In [92]: mix_list[2]
```

```
Out[92]: '5.0'
```

```
In [93]: mix_list[5]
```

```
-----  
IndexError                                Traceback (most recent call last)  
<ipython-input-93-efefc08ff73a> in <module>  
----> 1 mix_list[5]
```

```
In [95]: mix_list[-2]
```

```
Out[95]: 5.0
```

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

```
In [96]: elem_2 = mix_list[2]
```

```
In [97]: elem_2
```

```
Out[97]: '5.0'
```

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

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

```
In [98]: mix_list[2] = "A different string"
```

```
In [99]: mix_list
```

```
Out[99]: [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 [100]: len(mix_list)
```

```
Out[100]: 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 [101]: number_list = [423,34,1]
```

```
In [102]: sum(number_list)
```

```
Out[102]: 458
```

```
In [103]: min(number_list)
```

```
Out[103]: 1
```

```
In [104]: max(number_list)
```

```
Out[104]: 423
```

## How do we find the mean or average?

```
In [105]: sum(number_list)/len(number_list)
```

```
Out[105]: 152.66666666666666
```

```
In [ ]:
```

## Removing things from list

```
In [122]: number_list = [423,34,1, 423]
```

```
In [107]: number_list
```

```
Out[107]: [423, 34, 1, 423]
```

```
In [111]: number_list.remove(423)
```

```
In [114]: number_list
```

```
Out[114]: [423, 34, 1, 423]
```

```
In [121]: del number_list[1]
```

```
-----  
IndexError                                Traceback (most recent call last)  
<ipython-input-121-e11fdcf84ad1> in <module>  
----> 1 del number_list[1]  
  
IndexError: list assignment index out of range
```

```
In [120]: number_list
```

```
Out[120]: [423]
```

```
In [123]: number_list
```

```
Out[123]: [423, 34, 1, 423]
```

```
In [128]: number_list.append(27)
```

```
In [129]: number_list
```

```
Out[129]: [423, 34, 1, 423, 27, 27, 27]
```



# NumPy

"Numerical Python".

It's 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 [130]: import numpy as np
```

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

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 a lot like lists, but they are designed to work with numbers

Here is how we create one

```
In [131]: my_arr = np.array([1,2,3])
```

```
In [132]: my_arr
```

```
Out[132]: 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 [133]: my_arr*5
```

```
Out[133]: array([ 5, 10, 15])
```

```
In [134]: my_arr**3
```

```
Out[134]: array([ 1,  8, 27])
```

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

```
In [135]: num_list = [1,2,3]
```

```
In [136]: num_list*5
```

```
Out[136]: [1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]
```

```
In [137]: my_list**5
```

```
-----  
TypeError                                Traceback (most recent call last)  
<ipython-input-137-60150ab04e48> 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 [138]: np.mean(my_arr)
```

```
Out[138]: 2.0
```

```
In [ ]:
```