3. Types
What are types?

- Types are variables, containers, or placeholders which hold, or represent, different types of information (sometimes they can many different types).

- For example, if you want something to hold an integer it would be an *integer* type, i.e., \( a = 1 \). In this case you have assigned the value of 1, using the assignment operator “=” to a variable (operand) called “a”.

- Python contains numerous types and is very flexible in creating new types (objects; we will cover this later)
3. Python Types

What are the standard types (variables) available in Python?

- **Integer**
- **Float**
- **Bool**
- **Long** (these are simply long integers, i.e., you can store a bigger integer in memory)
- **Complex** (these are complex numbers, don’t worry about it for the moment)
- **String**
- **List**
- **Tuple**
- **Dictionary**
- **Sets**
- **File** (this will be covered in depth later)

In this course we will discuss into detail those shown in **bold**.
Integers are variables that contain an integer. Floats are variables that decimal or double precision values.

Operations on integers often have interesting behavior. Try this:

- \( a = 1 \)
- \( b = 2 \)
- \( a/b \)

Do you see what it is doing? This does not happen when you do the following:

- \( a = 1 \)
- \( b = 2.0 \)
- \( a/b \)
3. Python types
Integers and Floats

Integers can be incremented in the following ways:

- \( a = 1 \)
- \( a += 5 \)
- \( a = a + 5 \)

Define a float and try the above... You can combine the = operator with other mathematical operators.

- \( a = 1 \)
- \( a *= 10 \)
- \( a /= 10 \)
You can *cast* or convert between numerical types. For example try these commands which all return a new value and do not change the variable, `a`:

- `a = 1`
- `long(a)`
- `float(a)`
- `complex(a)`

You can even do the following:

- `a = 1`
- `str(a)`
3. Python types

A **Bool** is a numeric type that can take on two values, **True** and **False**. **True** and **False** are special Python variables. They are the same as 1 and 0, respectively. Try the following,

- `x = True`
- `y = False`
- `x is y`
- `x not y`
- `not(x) is y`
- `x is not(y)`
- `x or y`
- `x != y`
- `x == y`
3. Python types

Strings

String are assigned a value in the following way:

- `a = ' A lovely'

You can refer to individual elements (characters) from it:

- `a[1]`
- `a[0]`

*Note: strings (and lists) are indexed from 0 to n-1 in Python and most other languages.*

- `a[0:2]`

- `b=‘ world’`
- `a+b`
3. Python types

Strings

- Define two more variables, named left and right, containing the name of your left and your right neighbor in this room.

- Define a new string variable that is composed of the letters at uneven positions of their first names.

- Strings can also contain numbers. Define a string variable containing your telephone number.

- Get out the country code.
3. Python types
Strings - converting (and casting) strings

Strings can be converted to other types. Notable numeric types and lists (more on lists in a bit). Note that all of these methods return a value (i.e., not in place conversion).

Let’s create a string to work with...

- `s = '20'`

Converting a string to an integer.

- `a = int(s)`

Converting a string to a float.

- `f = float(s)`

Converting a string to a long integer.

- `l = long(s)`
Again, let’s create a new string to work with. Again all of these methods return a new string.

```
▶ s = 'Green beards'
```

Converting the string to upper case letters.

```
▶ a = s.upper()
▶ print a
```

Converting the string to lower case letters.

```
▶ a = s.lower()
▶ print a
```
3. Python types
Strings - converting (and casting) strings, continued

Capitalizing the first word.

- `a = s.capitalize()`
- `print a`

Swapping the case of the letters in a string.

- `a = s.swapcase()`
- `print a`
You can search a string for specific substrings.

Again, let’s create a new string to work with.
▶️ \( s = \text{‘The quick brown fox jumps over the lazy dog.’} \)

Let’s find the substring “fox”.
▶️ \( x = s.\text{find(‘fox’)} \)
▶️ \( \text{print } x \)

What do think the output means? Try this...
▶️ \( \text{print } s[x:x+\text{len}(s)] \)

Notes:
▶️ The method returns the index of the first letter of the first substring found (all others are ignored).
▶️ The method returns -1 if nothing matching is found.
Using the following string write some code that finds the indices (starting and ending) for of all occurrences of the word fox.

- \[ s = 'The quick brown fox jumps over the lazy fox.' \]

Answer...

```python
s = 'The quick brown fox jumps over the lazy fox.'
f = 'fox'
x = s.find(f, 0)
while x != -1:
    print(x, x+len(f))
    x = s.find(f, x+len(f))
```
There is another method that is nearly identical to find() for finding substrings within a string.

Again, let’s create a new string to work with.

```python
s = 'The quick brown fox jumps over the lazy dog.'
x = s.index('fox')
print x
```

Difference between index and find:

- The index() method returns a variable called `ValueError` rather than -1.

In all other respects they are similar in behavior.
3. Python types
String operations - finding substrings

There are a number of other substring search methods that we will only mention here:

- `rfind()`
- `rindex()`

The syntax for the use of the methods is identical. However, as might be apparent from their names they start their searches from the **right** and return the **highest index** of the first letter in the substring first.

For example, in the previous example that would be the index 40 corresponding the letter *f* in the second occurrence of *fox* in the sentence.
3. Python types
Strings - counting substrings (non-overlapping)

If you simply interested in determining the number of non-overlapping occurrence of substring you can use the `count()` method. For example,

```python
s = 'The quick brown fox jumps over the lazy dog.'
x = s.count('fox')
print x
```

As you can see the function returns the number of occurrences of `fox`. What is the result for the string `o`?
You can replace a substring with a new substring up to many occurrences as you wish (the method replaces from the left to the right).

- `s = 'The quick brown fox jumps over the lazy dog.'`
- `newStr = s.replace('fox', dog)`
- `print newStr`
- `newStr = s.replace('fox', dog, 1)`
- `print newStr`

What happens if you try to replace a substring that has zero occurrences?

- `s = 'The quick brown fox jumps over the lazy dog.'`
- `newStr = s.replace('hamster', dog)`
- `print newStr`

It returns a copy of the full string unaltered.
A list is an ordered sequence of elements, which can be of different types. This is a generic container to hold anything you want.

For example:

▶ `examplelist = ['bla', 10, 14, '+31']`

Referring to elements of a list is done in the same way as for strings:

▶ `examplelist[0]`

How would you refer to the 'a' of 'bla' in this list?
3. Python types

Lists

Lists can contain other lists as one of their elements (or any other Python type):

- `list2 = ['bla', [1, 2, 3, 'hello'], 10, 14, '+43']`

Indexing:

- `list2[0][3]`
- `list2[1][1]`
- `list2[1][3][4]`

You can ask for the length of a list with the function `len`:

- `len(list2)`

Does the returned value make sense?
3. Python types
Lists - membership operators

To check whether a list contains a particular element, use the operator `in`. Try entering:

- 10 in list2
- 7 in list2
- 'hello' in list2

The opposite of `in` is `not in`:

- 10 not in list2
- 7 not in list2
- 'hello' not in list2
3. Python types

List operations

To produce a list containing an ordered sequence of integers:

▶ Try `range(0,7,1)`

You can multiply a list with an integer, to create a list with additional copies of the contents (in order) of the original list:

▶ `m = [1, 2, 3, 'bla']`
▶ `n = l*3`

And you can add two lists together:

▶ `list2 + m`
You can add additional values as new elements using `append(new values)`.

- `myList = [1, 5, 'fred', 2.3456]`
- `myList.append('georgia')`

`append(new values)` only accepts a single new value that is added to the end of the list increasing its length by 1.
Extending a list

- Robert = [13, 678, 2]
- Robert.extend([1, 2, 3])
- print Robert

The extend method must be an iterable type. For example, this does not work

- Robert = [13, 678, 2]
- Robert.extend(4)
3. Python types
List operations - extending

But this does work,

- Robert = [13, 678, 2]
- Robert.extend([4])
- Robert.extend('green')
- print Robert

Notes the value ‘green’ is a string and EACH character is added as separate element in the list. Also, while extend iterates over the contents of the list to be added it does it only at the top level elements.
3. Python types
List operations - inserting and removing items

Inserting items.

This takes two arguments: the position where you want to add it (indexed 0 to n-1) and the item to add

- \( l = ['green', 1, 2, 3, 'red', 'green'] \)
- \( l.insert(2, 'orange') \)
- \( print(l) \)

Removing items

- \( l.remove('green') \)
- \( print(l) \)

What happens here?
There is another method for removing items from a list that is not based on the value of the item, but rather its position in the list.

- \( l = [\text{‘green’, 1, 2, 3, ‘red’, ‘green’}] \)
- \( l.pop(5) \)
- \( \text{print } l \)

What happens when you do this?

- \( l.pop() \)
- \( \text{print } l \)

Why do we have two ways of removing items from a list? The \texttt{pop()} method returns the value (which can then be assigned to a new variable) whereas the \texttt{remove} method does not. Try this...

- \( x = l.pop(0) \)
- \( \text{print } l \)
- \( \text{print } x \)
As with strings you can also count the number of occurrences of a type with a certain value found in a list.

```python
l = [1, 1, 23, 87, 4, 93, -1, 1, 3, 23]
x = l.count(1)
print x
```
3. Python types
List operations - finding the index (position) of an item in a list

Determining the index of items in a list is easy.

- `l = [1, 1, 23, 87, 4, 93, -1, 1, 3, 23]`
- `x = l.index(1)`
- `print x`

Note that `index()` returns the index of only the first item. Write some code that finds the indices of all of the elements that are 1 and uses the `pop()` method to remove them all.

```python
l = [1, 1, 23, 87, 4, 93, -1, 1, 3, 23]
x = None
while x != -1:
    try:
        x = l.index(1)
        l.pop(x)
        print 'The pop() method has removed index: ', x
    except:
        x = -1
print l
```