Python II

Built In Types

Simple values

Type	Example	Description
`int`	`x = 1`	Integers (i.e., whole numbers)
`float`	`x = 1.0`	Floating-point numbers (i.e., real numbers)
`complex`	`x = 1 + 2j`	Complex numbers (i.e., numbers with a real and imaginary part)
`bool`	`x = True`	Boolean: True/False values
`str`	`x = 'abc'`	String: characters or text
`NoneType`	`x = None`	Special object indicating nulls

Integers

- A number without a decimal point
- They are variable precision
- Division upcasts to floating point with division (Python 3)

```
In [1]:  x = 1
         type(x)

Out [1]: int
```

```
In [2]:  2 ** 200

Out [2]:
1606938044258990275541962092341162602522202993782792835301376
```

```
In [3]:  5 / 2

Out [3]: 2.5
```

Use // for floor division

```
In [4]:  5 // 2

Out [4]: 2
```

Floating point numbers

- Can be stored in scientific or standard decimal notation
- Ints can be converted to floats with the built in float() constructor

```
In [5]: x = 0.000005
        y = 5e-6
        print(x == y)

True

In [6]: x = 1400000.00
        y = 1.4e6
        print(x == y)

True


In [7]:  float(1)

Out [7]: 1.0
```

Floating point precision

- Since numbers are represented in binary in memory, floating points aren't completely precise
- Just as ⅓ can't be represented with a finite number of base 10 decimal places, 0.1 and other base 10 floating point numbers cannot be represented with a finite number of bits

```
In [8]:  0.1 + 0.2 == 0.3

Out [8]: False
```

```
In [9]: print("0.1 = {0:.17f}".format(0.1))
        print("0.2 = {0:.17f}".format(0.2))
        print("0.3 = {0:.17f}".format(0.3))
```

```
0.1 = 0.10000000000000001
0.2 = 0.20000000000000001
0.3 = 0.29999999999999999
```

$$1/3 = 0.333333333...$$

$$1/10 = 0.0001100110011001100..._2$$

String type

- Can be created with single or double quotes
- Some useful functions are on the right

```
In [18]:  # length of string
          len(response)

Out [18]: 4

In [19]:  # Make uppercase. See also str.lower()
          response.upper()

Out [19]: 'SPAM'

In [20]:  # Capitalize. See also str.title()
          message.capitalize()

Out [20]: 'What do you like?'

In [21]:  # concatenation with +
          message + response

Out [21]: 'what do you like?spam'

In [22]:  # multiplication is multiple concatenation
          5 * response

Out [22]: 'spamspamspamspamspam'

In [23]:  # Access individual characters (zero-based indexing)
          message[0]

Out [23]: 'w'
```

None type

- It has only a single possible value, None
- Used most commonly as a default return value for a function
- Try print() and see its return value

```
In [24]:  type(None)

Out [24]: NoneType


In [25]: return_value = print('abc')

abc

In [26]: print(return_value)

None
```

Boolean Type

A simple type with two possible values: True and False

They must be capitalized

```
In [27]:  result = (4 < 5)
          result

Out [27]: True

In [28]:  type(result)

Out [28]: bool
```

```
In [29]: print(True, False)

True False
```

Booleans continued

Booleans can be constructed using the `bool()` constructor

None converts to False

Empty strings are False, True otherwise

Empty sequences are False, otherwise True

```
In [30]: bool(2014)
Out [30]: True
In [31]: bool(0)
Out [31]: False
In [32]: bool(3.1415)
Out [32]: True

In [33]: bool(None)
Out [33]: False

In [34]: bool("")
Out [34]: False
In [35]: bool("abc")
Out [35]: True

In [36]: bool([1, 2, 3])
Out [36]: True
In [37]: bool([])
Out [37]: False
```

Built-in Data Structures

Type Name	Example	Description
`list`	`[1, 2, 3]`	Ordered collection
`tuple`	`(1, 2, 3)`	Immutable ordered collection

Type Name	Example	Description
`dict`	`{'a':1, 'b':2, 'c':3}`	Unordered (key,value) mapping
`set`	`{1, 2, 3}`	Unordered collection of unique values

Lists

- Lists are ordered and mutable and can be of any type or mixed types, defined with syntax below

```
In [1]: L = [2, 3, 5, 7]
```

- Some useful properties are on the right see below for more

https://docs.python.org/3/tutorial/datastructures.html

```
In [2]: # Length of a list
        len(L)

Out [2]: 4

In [3]: # Append a value to the end
        L.append(11)
        L

Out [3]: [2, 3, 5, 7, 11]

In [4]: # Addition concatenates lists
        L + [13, 17, 19]

Out [4]: [2, 3, 5, 7, 11, 13, 17, 19]

In [5]: # sort() method sorts in-place
        L = [2, 5, 1, 6, 3, 4]
        L.sort()
        L

Out [5]: [1, 2, 3, 4, 5, 6]
```

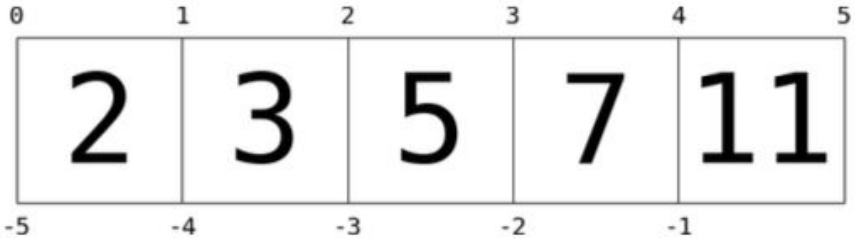
List indexing and slicing

- The syntax for indexing is on the right
- Syntax for slicing is below

```
In [7]: L = [2, 3, 5, 7, 11]
```

```
In [8]:  L[0]
Out [8]: 2
In [9]:  L[1]
Out [9]: 3
```

```
In [10]:  L[-1]
Out [10]: 11
In [12]:  L[-2]
Out [12]: 7
```




```
In [12]:  L[0:3]
Out [12]: [2, 3, 5]
```

```
In [13]:  L[:3]
Out [13]: [2, 3, 5]
```

```
In [13]:  L[:3]
Out [13]: [2, 3, 5]
```

```
In [13]:  L[:3]
Out [13]: [2, 3, 5]
```