Python Data Types (2): Lists and Tuples

- Lists
- Tuples
- Objects and Classes
- Python Standard Library

Lists

We have already seen the Python string data type, which is a sequence of characters enclosed in quote marks. E.g. "hello, world!"

Python also has a more general sequence type – a list. A list is a comma-separated sequence of items enclosed within square brackets.

```
[0, 1, 'two', 'three', [4, 'five']]
```

The items in a list can be any type – numbers, strings, and even other lists

```
>>> pets = ['ant', 'bat', 'cod', 'dog', 'elk']
>>> lst = [0, 1, 'two', 'three', [4, 'five']]
>>> nums = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

List operators and functions

Like strings, lists can be manipulated with operators and functions

Usage	Explanation
`x in lst`	`x` is an item of `lst`
`x not in lst`	`x` is not an item of `lst`
`lst + lstB`	Concatenation of `lst` and `lstB`
`lst*n, n*lst`	Concatenation of `n` copies of `lst`
`lst[i]`	Item at index `i` of `lst`
`len(lst)`	Number of items in `lst`
`min(lst)`	Minimum item in `lst`
`max(lst)`	Maximum item in `lst`
`sum(lst)`	Sum of items in `tple`

```
>>> lst = [1, 2, 3]
>>> lstB = [0, 4]
>>> 4 in lst
False
>>> 4 not in lst
True
>>> lst + lstB
[1, 2, 3, 0, 4]
>>> 2*lst
[1, 2, 3, 1, 2, 3]
>>> lst[0]
1
>>> lst[1]
2
>>> lst[-1]
3
>>> len(lst)
3
>>> min(lst)
1
>>> max(lst)
3
>>> sum(lst)
6
>>> help(list)
...
```

Lists are mutable, strings are not

Lists can be modified; they are said to be mutable

```
pets = ['ant', 'bat', 'cow', 'dog', 'elk']
```

Strings can't be modified; they are said to be immutable

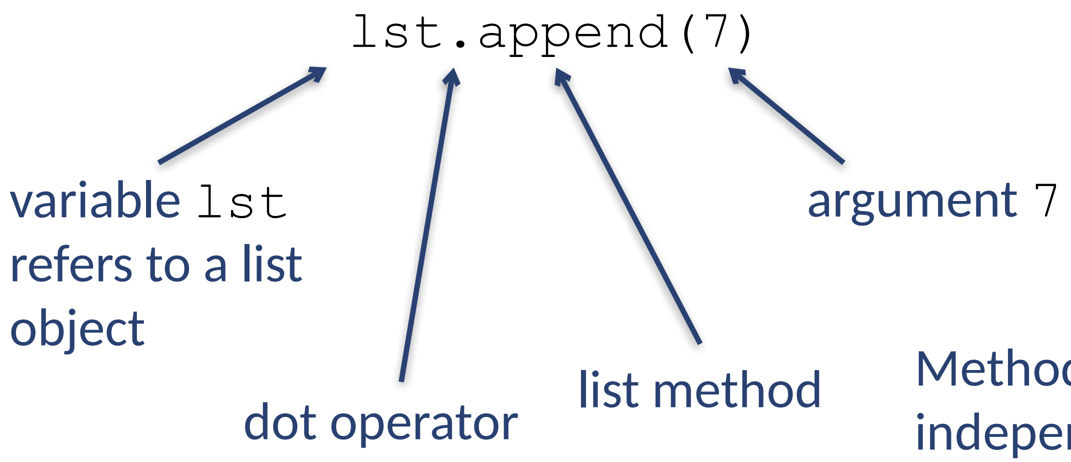
```
pet = 'cod'
```

```
>>> pets = ['ant', 'bat', 'cod', 'dog', 'elk']
>>> pets[2] = 'cow'
>>> pets
['ant', 'bat', 'cow', 'dog', 'elk']
>>> pet = 'cod'
>>> pet[2] = 'w'
Traceback (most recent call last):
  File "<pyshell#155>", line 1, in <module>
    pet[2] = 'w'
TypeError: 'str' object does not support item assignment
>>>
```

List methods

`len()` and `sum()` are examples of functions that can be called with a list input argument; they can also be called on other types of input

There are also functions that are called on a list; such functions are called list methods



```
>>> lst = [1, 2,
3]
>>> len(lst)
3
>>> sum(lst)
6
>>> lst.append(7)
>>> lst
[1, 2, 3, 7]
>>>
```

Method `append()` can't be called independently; it must be called on some list object with the dot operator

Lists methods

Usage	Explanation
`lst.append(item)`	adds `item` to the end of `lst`
`lst.count(item)`	returns the number of times `item` occurs in `lst`
`lst.index(item)`	Returns index of (first occurrence of) `item` in `lst`
`lst.pop( )`	Removes and returns the last item in `lst`
`lst.remove(item)`	Removes (the first occurrence of) `item` from `lst`
`lst.reverse( )`	Reverses the order of items in `lst`
`lst.sort( )`	Sorts the items of `lst` in increasing order

Methods `append()`, `remove()`, `reverse()`, `sort()` modify `lst` but do not return the modified list

```
>>> lst = [1,2,3]
>>> lst.append(7)
>>> lst.append(3)
>>> lst
[1, 2, 3, 7, 3]
>>> lst.count(3)
2
>>> lst.remove(2)
>>> lst
[1, 3, 7, 3]
>>> lst.reverse()
>>> lst
[3, 7, 3, 1]
>>> lst.index(3)
0
>>> lst.sort()
>>> lst
[1, 3, 3, 7]
>>> lst.remove(3)
>>> lst
[1, 3, 7]
>>> lst.pop()
7
>>> lst
[1, 3]
```

Exercise

List `lst` is a list of prices for a pair of boots at different online retailers

a) You found another retailer selling the boots for $160.00; add this price to `lst`
b) Compute the number of retailers selling the boots for $160.00
c) Find the minimum price in `lst`
d) Using c), find the index of the minimum price in `lst`
e) Using c) remove the minimum price from list `lst`
f) Sort `lst` in increasing order

```
>>> lst = [159.99, 160.00, 205.95,
128.83, 175.49]
>>> lst.append(160.00)
>>> lst.count(160.00)
2
>>> min(lst)
128.83
>>> lst.index(128.83)
3
>>> lst.remove(128.83)
>>> lst
[159.99, 160.0, 205.95, 175.49, 160.0]
>>> lst.sort()
>>> lst
[159.99, 160.0, 160.0, 175.49, 205.95]
>>>
```

Tuples

Python also has a general sequence type that is similar to a list, but is immutable. A tuple is a comma-separated sequence of items enclosed in parentheses.

```
(0, 1, 'two', 'three', [4, 'five'])
```

The items in a tuple can be any type – numbers, strings, and even other tuples or lists

The parentheses around the tuple may be omitted, but it is customary and clearer to include them.

```
>>> pets = ('ant', 'bat', 'cod', 'dog', 'elk')
>>> tple = (0, 1, 'two', 'three', [4, 'five'])
>>> nums = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
>>> noParens = 'a', 'b'
>>> type(noParens)
<class 'tuple'>
```

Tuple operators and functions

Tuple methods **may not** change the tuple. `pop`, `reverse`, `delete`, `append`, `remove`, `reverse` and `sort` are therefore **not** tuple methods.

Usage	**Explanation**
`x in tple`	`x` is an item of `tple`
`x not in tple`	`x` is not an item of `tple`
`tpleA + tpleB`	Concatenation of `tpleA` and `lstB`
`tple*n`, `n*tple`	Concatenation of `n` copies of `tple`
`tple[i]`	Item at index `i` of `tple`
`len(tple)`	Number of items in `tple`
`min(tple)`	Minimum item in `tple`
`max(tple)`	Maximum item in `tple`

```
>>> tpleA = (1, 2,
3)
>>> tpleB = (0, 4)
>>> 4 in tpleA
False
>>> 4 not in tpleA
True
>>> tpleA+ tpleB
(1, 2, 3, 0, 4)
>>> 2*tple
(1, 2, 3, 1, 2, 3)
>>> tple [0]
1
>>>
tpleA.append('C')
Traceback (most
recent call last):
  File
"<pyshell#15>", line
1, in <module>

tpleA.append('C')
AttributeError:
'tuple' object has
no attribute
'append' >>>
len(tpleA)
3
>>> min(tpleA)
1
```

Slicing

While indexing is used to obtain individual elements, slicing allows you to obtain a specified range of sequence's elements (i.e., list, tuple, and str).

The simplest slice syntax is as follows:

```
lst[start:stop]
```

`lst` refers to a sequence of type list.

Starting index of the slice. Defaults to 0.

Last index of the slice. Defaults to len(lst).

```
>>> lst = [1, 3, 5, 7]
>>> lst[1:4]
[3, 5, 7]
>>> lst[:2]
[1, 3]
>>>
```

- The *start* and *stop* values are optional. When missing, Python uses the default values.
- The slice includes the element at index *start*, but does not include the element at index *stop*.

Exercise

Given the two assignments:

```
p1 = ['do', 'it', 'better']
p2 = ['make', 'us', 'stronger']
```

Create expressions that perform the following:

a) Use slicing to obtain the second element in `p1` (i.e., the element at index 1)

b) Using slicing and the lists p1 and p2, make a new list named `lst` with the following elements:

```
['make', 'it', 'better']
```

Exercise

Given the two assignments:

```
p1 = ['do', 'it', 'better']
p2 = ['make', 'us', 'stronger']
```

Create expressions that perform the following:

a) Use slicing to obtain the second element in `p1` (i.e., the element at index 1)

b) Using slicing and the lists p1 and p2, make a new list named `lst` with the following elements:
`['make', 'it', 'better']`

```
>>> p1[1:2]
['it']
>>> lst = p2[:1] + p1[-2:]
>>> lst
['make', 'it', 'better']
>>>
```

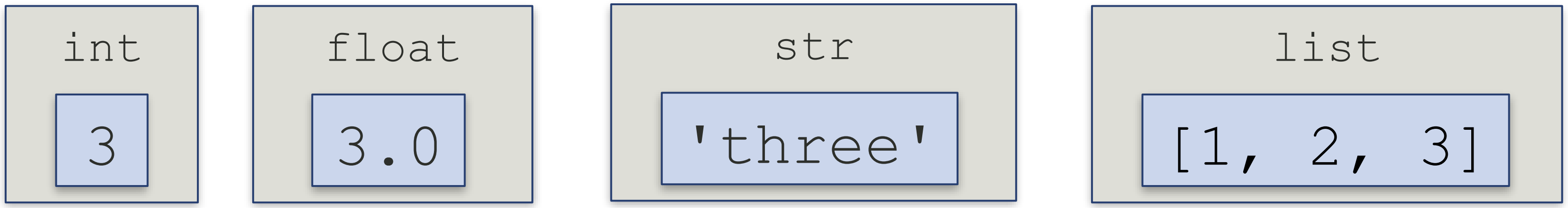
Objects and classes

Let's summarize what we know about objects and classes in Python.

In Python, every value, whether a simple integer value like `3` or a more complex value, such as the list `['hello', 4,  5]` is stored in memory as an object.

Every object has a value and a type.

```
>>> a = 3
>>> b = 3.0
>>> c = 'three'
>>> d = [1, 2, 3]
>>> type(a)
<class 'int'>
>>> type(b)
<class 'float'>
>>> type(c)
<class 'str'>
>>> type(d)
<class 'list'>
>>> a = []
>>> type(a)
<class 'list'>
```



An object’s type determines what values it can have and how it can be manipulated

Terminology: object X is of type `int` = object X belongs to class `int`

Values of number types

An object's type determines what values it can have and how it can be manipulated

An object of type `int` can have, essentially, any integer number value

The value of an object of type `float` is represented in memory using 64 bits

- i.e., 64 zeros and ones

This means that only 2^{64} values can be represented with a `float`; any other real number value must be approximated by one of these values. A value that is too big causes an Overflow Error.

```
>>> 0
0
>>> 2**1024
1797693134862315907729305
1907890247336179769789423
0657273430081157732675805
5009631327084773224075360
2112011387987139335765878
9768814416622492847430639
4741243777678934248654852
7630221960124609411945308
2952085005768838150682342
4628814739131105408272371
6335051068458629823994724
5938479716304835356329624
224137216
>>> 0.0
0.0
>>> 2.0**1024
Traceback (most recent
call last):
  File "<pyshell#38>",
line 1, in <module>
    2.0**1024
OverflowError: (34,
'Result too large')
>>> 2.0**(-1075)
0.0
```

Operators for number types

An object's type determines what values it can have and how it can be manipulated

We already saw the operators that are used to manipulate number types

- algebraic operators +, −, `*`, `/`, `//`, `%`, `**`, `abs()`
- comparison operators >, <, ==, `!=`, <=, >=, …

Parentheses and precedence rules determine the order in which operators are evaluated in an expression

higher precedence ↑

lower precedence ↓

Operator
`[…]`
`x[]`
`**`
`+x, -x`
`*, /, //, %`
`+, -`
`in, not in`
`<,>,<=,>=,==,!=`
`not x`
`and`
`or`

Object constructors

An assignment statement can be used to create an integer object with value 3

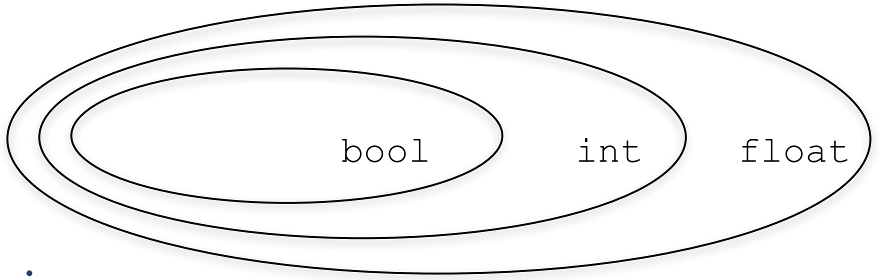
- The type of the object is implicitly defined by the type of the value

The object can also be created by explicitly specifying the object type using a constructor function

- `int()`: integer constructor (default value: `0`)
- `float()`: Float constructor (default value: `0.0`)
- `str()`: string constructor (default value: empty string `''`)
- `list()`: list constructor (default value: empty list `[]`)

```
>>> x = 3
>>> x
3
>>> x = int(3)
>>> x
3
>>> x = int()
>>> x
0
>>> y = float()
>>> y
0.0
>>> s = str()
>>> s
''
>>> lst = list()
>>> lst
[]
>>>
```

Type conversion



Implicit type conversion

- When evaluating an expression that contains operands of different type, operands must first be converted to the same type
- Operands are converted to the type that "contains the others"

Explicit type conversion

- Constructors can be used to explicitly convert types

`int()` creates an `int` object

- from a `float` object, by removing decimal part
- from a `str` object, if it represents an integer

`float()` creates a `float` object

- from an `int` object, if it is not too big
- from a `string`, if it represents a number

`str()` creates a `str` object

- the string representation of the object value

```
>>> str(345)
'345'
>>> str(34.5)
'34.5'
>>> int(2.1)
2
>>> int('456')
456
>>> float('45.6')
45.6
>>> float(2**100)
1267650600228229401496703205376.0
>>> 2 + 3.0
5.0
```

Classes and methods

In Python, every value is stored in memory as an object. Every object belongs to a class (i.e., has a type). The object's class determines what operations can be performed on it.

We saw the operations that can be performed on classes `int` and `float`

The `list` class supports:

- operators such as `+`, `*`, `in`, `[]`
- methods such as `append()`, `count()`, `remove()`, `reverse()`

```
>>> fish = ['goldfish']
>>> fish * 2
['goldfish', 'goldfish']
>>> myPets = ['cat', 'dog']
>>> pets = fish + myPets
>>> pets
['goldfish', 'cat', 'dog']
>>> 'frog' in pets
False
>>> pets[-1]
'dog'
>>> pets.append('guinea pig')
>>> pets.append('dog')
>>> pets
['goldfish', 'cat', 'dog',
'guinea pig', 'dog']
>>> pets.count('dog')
2
>>> pets.remove('dog')
>>> pets
['goldfish', 'cat', 'guinea
pig', 'dog']
>>> pets.reverse()
>>> pets
['dog', 'guinea pig', 'cat',
'goldfish']
```

Python Standard Library

The standard Python distribution comes with builtin functions such as `max()` and `sum()` and classes such as `int`, `str`, and `list`.

Many more functions and classes are defined in the Python Standard Library to support

- Network programming
- Web application programming
- Graphical user interface (GUI) development
- Database programming
- Mathematical functions
- Pseudorandom number generators
- Media processing, etc.

The Python Standard Library functions and classes are organized into components called modules.

Standard Library module `math`

The core Python language does not have a square root function

The square root function `sqrt()` is defined in the Standard Library module `math`

A module must be explicitly imported into the execution environment:

```
import <module>
```

The prefix `math` must be used to access the function `sqrt()`

The `math` module is a library of mathematical functions and constants

```
>>> import math
>>> math.sqrt(4)
2.0
>>> help(math)
Help on module math:
…
>>> math.cos(0)
1.0
>>> math.log(8)
2.0794415416798357
>>> math.log(8, 2)
3.0
>>> math.pi
3.141592653589793
```

Exercise

Write a Python expression that assigns to variable c

a) The length of the hypotenuse in a right triangle whose other two sides have lengths 3 and 4
b) The value of the Boolean expression that evaluates whether the length of the above hypotenuse is 5
c) The area of a disk of radius 10
d) The value of the Boolean expression that checks whether a point with coordinates (5, 5) is inside a circle with center (0,0) and radius 7.

```
>>> c = math.sqrt(3**2+4**2)
>>> c
5.0
>>> c = (math.sqrt(3**2+4**2)
== 5)
>>> c
True
>>> c = math.pi*10**2
>>> c
314.1592653589793
>>> c = (2*5**2 < 7**2)
>>> c
False
```