Basic Programming

Lesson 3. Collections: Lists, Tuples

Alexandra Ciobotaru

29 oct 2021

Syllabus

- **Lesson 1. Computers, Programming and Cognitive Science. From pseudocode to programming languages.**
- **Lesson 2. Variables in Python. Basic calculus. Using Math library, type() and help() functions.**
- **Lesson 3. Collections: Lists, Tuples.**
- Lesson 4. Strings. Working with strings.
- Lesson 5. Branching and decisions: Logical operators, If-Statements, Nested conditions. Loops: For and While
- Lesson 6. Working with matrices in Python. Python numpy library. Quiz 1 (25%).
- Lesson 7. Collections: Dictionaries. JSON construction.
- Lesson 8. Working with files. Reading and Writing.
- Lesson 9. Analyzing dataframes. Pandas and Matplotlib Python libraries.
- Lesson 10. Creating functions. Recursive functions. Quiz 2 (25%).
- Lesson 11. Object-Oriented Programming: Encapsulation, Inheritance and Polymorphism.
- Lesson 12. Object-Oriented Programming (cont.). Error handling. Best practices when programming.
- Lab (20%) + final exam (30%).

I. Lists (https://www.youtube.com/watch?v=ohCDWZgNIU0)

- As opposed to data types such as int, bool, float, str, a list is a compound data type where you can group values together.
- It would be inconvenient to create a new python variable for each data point you use in your code. Instead, you could store all variables in a python list.
- Lists are defined in Python by enclosing a comma-separated sequence of objects in square brackets ([]), as shown below:
- my_list = [‘a’, ‘b’, ‘c’]

Characteristics of Python lists

1. Lists are ordered.
2. Lists can contain any arbitrary objects.
3. List elements can be accessed by index.
4. Lists can be nested to arbitrary depth.
5. Lists are mutable.
6. Lists are dynamic.

1. Lists are ordered

- A list is an ordered collection of objects
- Lists that have the same elements in a different order are not the same:
  - >>> a = ['foo', 'bar', 'baz', 'qux']
  - >>> b = ['baz', 'qux', 'bar', 'foo']
  - >>> a == b
  - False
  - >>> a is b
  - False

  - >>> [1, 2, 3, 4] == [4, 1, 3, 2]
  - False

2. Lists Can Contain Arbitrary Objects

- A list can contain any assortment of objects.
- The elements of a list can all be the same type:
  - >>> a = [2, 4, 6, 8]
  - >>> a
  - [2, 4, 6, 8]
- Or the elements can be of varying types:
  - >>> a = [21.42, 'foobar', 3, 4, 'bark', False, 3.14159]
  - >>> a
  - [21.42, 'foobar', 3, 4, 'bark', False, 3.14159]
- List objects needn’t be unique. A given object can appear in a list multiple times:
  - a = ['bark', 'meow', 'woof', 'bark', 'cheep', 'bark']

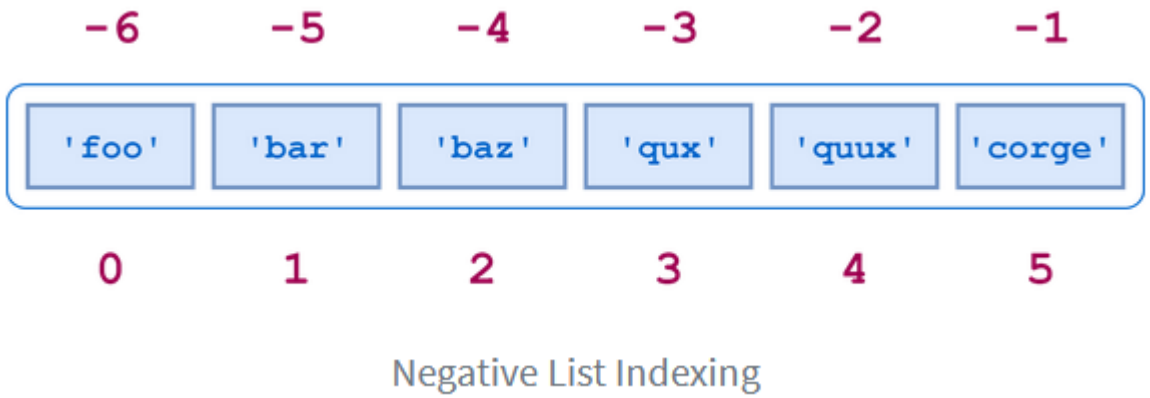
3. List Elements Can Be Accessed by Index

- Individual elements in a list can be accessed using an index in square brackets.
- If we have the list:
  - a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']
- The indices for the elements are:
- To access elements in list a:
  - >>> a[0]
  - 'foo'
  - >>> a[2]
  - 'baz'
  - >>> a[5]
  - 'corge'

Image source: https://realpython.com/python-lists-tuples/

Indexing in a negative way

- >>> a[-1]
- 'corge'
- >>> a[-2]
- 'quux'
- >>> a[-5]
- 'bar'



Negative List Indexing

Image source: https://realpython.com/python-lists-tuples/

Slicing a list

- Slicing means extracting only a ‘slice’ of your list
- a[m:n] returns the portion of a from index m to, but not including, index n:
  - >>> a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']
  - >>> a[2:5]
  - ['baz', 'qux', 'quux']
- Negative slicing is also a thing:
  - >>> a[-5:-2]
  - ['bar', 'baz', 'qux']
  - >>> a[1:4]
  - ['bar', 'baz', 'qux']
  - >>> a[-5:-2] == a[1:4]
  - True

Slicing a list by omitting first / second index

- Ommiting the first index starts the slice at the beginning of the list
  - >>> a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']
  - >>> a[:4]
  - ['foo', 'bar', 'baz', 'qux']
- Ommiting the second index extends the slice to the end of the list:
  - >>> a[4:]
  - ['quux', 'corge']

Strides and list reversing

- You can specify a stride, either positive or negative, to skip some elements in list:
  - >>> a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']
  - >>> a[0:6:2]
  - ['foo', 'baz', 'quux']
  - >>> a[1:6:2]
  - ['bar', 'qux', 'corge']
  - >>> a[6:0:-2]
  - ['corge', 'qux', 'bar']
- To reverse the list you can type:
  - >>> a[::-1]
  - ['corge', 'quux', 'qux', 'baz', 'bar', 'foo']

Other operations on lists

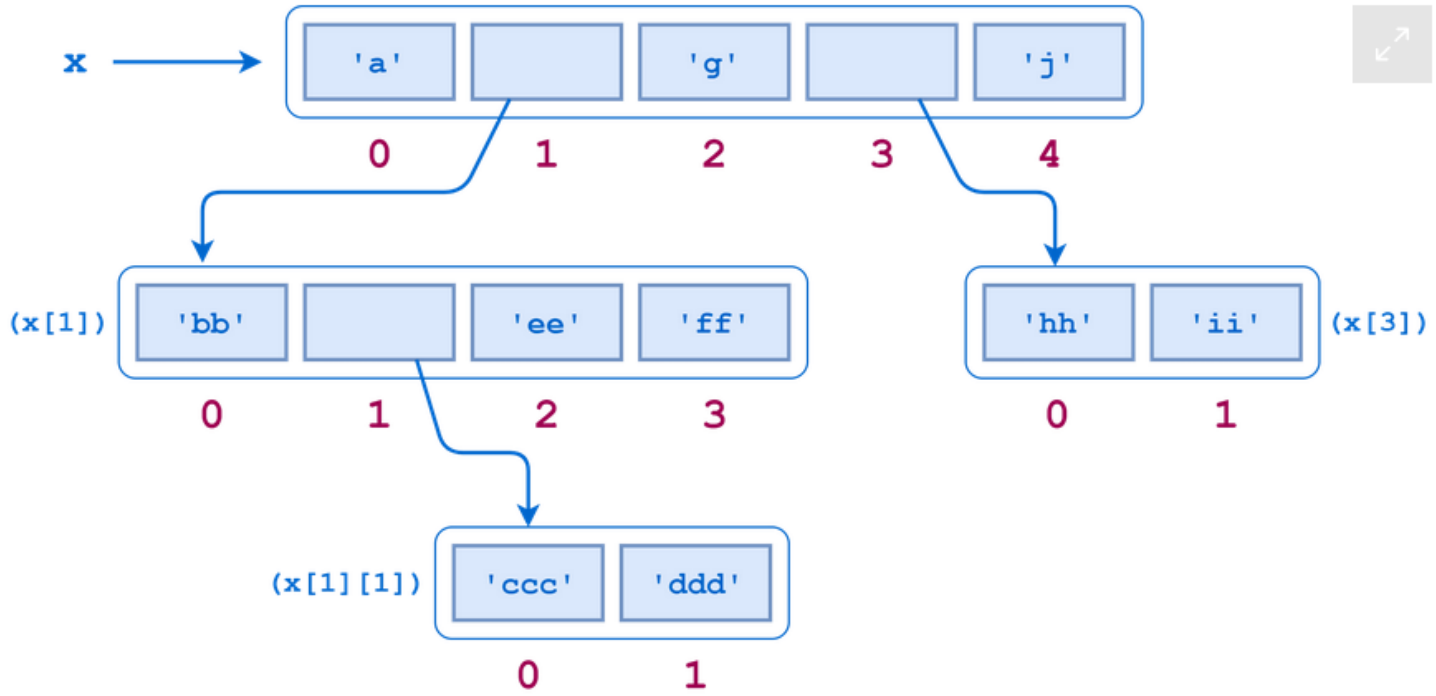
- To find out if an element is in a list or not:
  - a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']
  - >>> 'qux' in a
  - True
  - >>> 'thud' not in a
  - True
- Concatenation and replication:
  - >>> a + ['grault', 'garply']
  - ['foo', 'bar', 'baz', 'qux', 'quux', 'corge', 'grault', 'garply']
  - >>> a * 2
  - ['foo', 'bar', 'baz', 'qux', 'quux', 'corge', 'foo', 'bar', 'baz',
  - 'qux', 'quux', 'corge']

Len, min and max

- a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']
- Finding the length of a list:
  - >>> len(a)
  - 6
- Finding the minimum in a list:
  - >>> min(a)
  - 'bar'
- Finding the maximum in a list:
  - >>> max(a)
  - 'qux'

4. Lists can be nested to arbitrary depth

- x = ['a', ['bb', ['ccc', 'ddd'], 'ee', 'ff'], 'g', ['hh', 'ii'], 'j']



A Nested List

Image source: https://realpython.com/python-lists-tuples/

x[0], x[2], and x[4] are strings, each one character long:

```
>>> print(x[0], x[2], x[4])
a g j
```

But x[1] and x[3] are sublists:

```
>>> x[1]
['bb', ['ccc', 'ddd'], 'ee', 'ff']
>>> x[3]
['hh', 'ii']
```

Accessing items of nested lists

- To access the items in a sublist, simply append an additional index:
  - >>> x[1]
  - ['bb', ['ccc', 'ddd'], 'ee', 'ff']

  - >>> x[1][0]
  - 'bb'
  - >>> x[1][1]
  - ['ccc', 'ddd']
  - >>> x[1][2]
  - 'ee'
  - >>> x[1][3]
  - 'ff'

  - >>> x[3]
  - ['hh', 'ii']
  - >>> print(x[3][0], x[3][1])
  - hh ii

5. Lists are mutable

- Once a list has been created, elements can be added, deleted, shifted, and moved around at will. Python provides a wide range of ways to modify lists.
- A single value in a list can be replaced by indexing and simple assignment:
  - >>> a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']
  - >>> a[2] = 10
  - >>> a[-1] = 20
  - >>> a
  - ['foo', 'bar', 10, 'qux', 'quux', 20]
- You can delete values from lists:
  - >>> del a[3]
  - >>> a
  - ['foo', 'bar', 'baz', 'quux', 'corge']

.append() and .extend() methods

- With .append(), you can put at the end of a list a single element:
  - >>> a = ['a', 'b']
  - >>> x = a.append(123)
  - >>> print(x)
  - None
  - >>> a
  - ['a', 'b', 123]
- With .extend(), you can multiple elements at the end of a list:
  - >>> a = ['a', 'b']
  - >>> a.extend([1, 2, 3])
  - >>> a
  - ['a', 'b', 1, 2, 3]

.insert(), .remove() and .pop()

- a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']

- .**insert**(<index>,<obj>) inserts an object into a list, at the specified index
  - >>> a.insert(3, 3.14159)
  - >>> a
  - ['foo', 'bar', 'baz', 3.14159, 'qux', 'quux', 'corge']
- .**remove**(<obj>) removes the specified object from the list
  - a.remove('baz')
  - >>> a
  - ['foo', 'bar', 3.14159, 'qux', 'quux', 'corge']
- .**pop**(<index>) removes and returns the object at the index specified
  - >>> a.pop() (if no index is specified, it removes the last element)
  - >>> 'corge'
  - >>> a
  - ['foo', 'bar', 3.14159, 'qux', 'quux']
  - >>> a.pop(2)
  - >>> 3.14159
  - >>> a
  - ['foo', 'bar', 'qux', 'quux‘]

6. Lists are dynamic

- Meaning that they can adjust after the addition/removal of an object inside them.
- When items are added to a list, it grows as needed, and similarly, a list shrinks to accommodate the removal of items.

- To see all methods applicable to a list, type:
  - >>> dir(a)
- To find out what a specific method does to your list and how to use it, type:
  - >>> help(a.reverse)

II. Tuples (https://www.youtube.com/watch?v=NI26dqhs2Rk)

- Tuples are identical to lists in all respects, except for the following properties:
  - Tuples are defined by enclosing the elements in parentheses (()) instead of square brackets ([]).
  - Tuples are **immutable (= cannot be changed)**.
- Even being defined using parentheses, you can still index and slice tupkes using square brakets, just as for lists.
- Everything you've learned about lists—they are ordered, they can contain arbitrary objects, they can be indexed and sliced, they can be nested—is true of tuples as well. But **they can't be modified**:
  - >>> t = ('foo', 'bar', 'baz', 'qux', 'quux', 'corge')
  - >>> t[2] = 'Bark!'
  - Traceback (most recent call last):
  - File "<pyshell#65>", line 1, in <module>
  - t[2] = 'Bark!'
  - TypeError: 'tuple' object does not support item assignment

Why use a tuple instead of a list?

- Program execution is faster when manipulating a tuple than it is for the equivalent list. (This is probably not going to be noticeable when the list or tuple is small.)
- Sometimes you don’t want data to be modified. If the values in the collection are meant to remain constant for the life of the program, using a tuple instead of a list guards against accidental modification.

  - >>> a = [1,2,3,4,5]
  - >>> dir(a) -> append, clear, copy, count, extend, index, insert, pop, remove, reverse, sort (lists occupy more memory than tuples)
  - >>> a = (1,2,3,4,5)
  - >>> dir(a) -> count, index

Homework

- Compute and compare the size of a list (in bytes) versus the same list as a tuple:

```
import sys
a = (1,2,3,4,5)
b = [1,2,3,4,5]
print(sys.getsizeof(a))
>>> 80
print(sys.getsizeof(b))
>>> 96
```

- Compute and compare the execution time of creating a list 1 million times versus creating a tuple 1 million times:

```
import timeit
list_time = timeit.timeit(stmt = "[1,2,3,4,5]",
number = 1000000)
tuple_time = timeit.timeit(stmt = "(1,2,3,4,5)",
number = 1000000)
list_time
>>> 0.04277949999959674
tuple_time
>>> 0.007052200000543962
```

Thank you!