Summer Coding School 2023

Aug 12, 2023

Taehee Jeong, Ph.D.

Last week’s Agenda

Data type

- Integer
- Floating point
- String
- Boolean

Python Challenge2

- Reverse Words in a Sentence
- "Hello World“ → "World Hello"

Source: https://flatironschool.com, Python Popularity by Anna Johnson

Today's Agenda

Data structure

- List
- Set
- Tuples
- Dictionary

What is Not a “Collection”?

Most of our variables have one value in them - when we put a new value in the variable, the old value is overwritten.

```
$ python
>>> x = 2
>>> x = 4
>>> print(x)
4
```

A List is a Kind of Collection

A collection allows us to put many values in a single “variable”.

A collection is nice because we can carry all many values around in one convenient package.

```
friends = [ 'Joseph', 'Glenn', 'Sally' ]

carryon = [ 'socks', 'shirt', 'perfume' ]
```

List Constants

List constants are surrounded by square brackets and the elements in the list are separated by commas

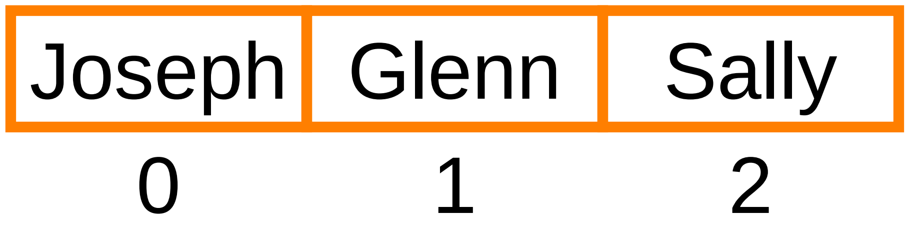
A list element can be any Python object - even another list

A list can be empty

```
>>> print([1, 24, 76])
[1, 24, 76]
>>> print(['red', 'yellow', 'blue'])
['red', 'yellow', 'blue']
>>> print(['red', 24, 98.6])
['red', 24, 98.6]
>>> print([ 1, [5, 6], 7])
[1, [5, 6], 7]
>>> print([])
[]
```

Looking Inside Lists

Just like strings, we can get at any single element in a list using an index specified in square brackets



```
>>> friends = [ 'Joseph', 'Glenn', 'Sally' ]
>>> print(friends[1])
Glenn
```

Lists are Mutable

Strings are “immutable” - we cannot change the contents of a string - we must make a new string to make any change

Lists are “mutable” - we can change an element of a list using the index operator

```
>>> fruit = 'Banana'
>>> fruit[0] = 'b'
Traceback
TypeError: 'str' object does not
support item assignment
>>> x = fruit.lower()
>>> print(x)
banana
>>> lotto = [2, 14, 26, 41, 63]
>>> print(lotto)
[2, 14, 26, 41, 63]
>>> lotto[2] = 28
>>> print(lotto)
[2, 14, 28, 41, 63]
```

How Long is a List?

The len() function takes a list as a parameter and returns the number of elements in the list

Actually len() tells us the number of elements of any set or sequence (such as a string...)

```
>>> greet = 'Hello Bob'
>>> print(len(greet))
9
>>> x = [ 1, 2, 'joe', 99]
>>> print(len(x))
4
```

Using the range Function

The range function returns a list of numbers that range from zero to one less than the parameter

We can construct an index loop using for and an integer iterator

```
>>> list(range(4))
[0, 1, 2, 3]
>>> friends = ['Joseph', 'Glenn', 'Sally']
>>> print(len(friends))
3
>>> list(range(len(friends)))
[0, 1, 2]
```

Concatenating Lists Using +

We can create a new list by adding two existing lists together

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> print(c)
[1, 2, 3, 4, 5, 6]
>>> print(a)
[1, 2, 3]
```

Lists Can Be Sliced Using :

```
>>> t = [9, 41, 12, 3, 74, 15]
>>> t[1:3]
[41,12]
>>> t[:4]
[9, 41, 12, 3]
>>> t[3:]
[3, 74, 15]
>>> t[:]
[9, 41, 12, 3, 74, 15]
```

Building a List from Scratch

We can create an empty list and then add elements using the append method

The list stays in order and new elements are added at the end of the list

```
>>> stuff = list()
>>> stuff.append('book')
>>> stuff.append(99)
>>> print(stuff)
['book', 99]
>>> stuff.append('cookie')
>>> print(stuff)
['book', 99, 'cookie']
```

Is Something in a List?

Python provides two operators that let you check if an item is in a list

These are logical operators that return True or False

They do not modify the list

```
>>> some = [1, 9, 21, 10, 16]
>>> 9 in some
True
>>> 15 in some
False
>>> 20 not in some
True
```

Lists are in Order

A list can hold many items and keeps those items in the order until we do something to change the order

A list can be sorted (i.e., change its order)

The sort method (unlike in strings) means “sort yourself”

```
>>> friends = [ 'Joseph', 'Glenn', 'Sally' ]
>>> friends.sort()
>>> print(friends)
['Glenn', 'Joseph', 'Sally']
>>> print(friends[1])
Joseph
```

Built-in Functions and Lists

There are a number of functions built into Python that take lists as parameters

```
>>> nums = [3, 41, 12, 9, 74, 15]
>>> print(len(nums))
6
>>> print(max(nums))
74
>>> print(min(nums))
3
>>> print(sum(nums))
154
>>> print(sum(nums)/len(nums))
25.6
```

Strings and Lists

```
>>> abc = 'With three words'
>>> stuff = abc.split()
>>> print(stuff)
['With', 'three', 'words']
>>> print(len(stuff))
3
>>> print(stuff[0])
With
>>> print(stuff)
['With', 'three', 'words']
```

Split breaks a string into parts and produces a list of strings. We think of these as words. We can access a particular word or loop through all the words.

```
>>> line = 'A lot               of spaces'
>>> etc = line.split()
>>> print(etc)
['A', 'lot', 'of', 'spaces']
>>>
>>> line = 'first;second;third'
>>> thing = line.split()
>>> print(thing)
['first;second;third']
>>> print(len(thing))
1
>>> thing = line.split(';')
>>> print(thing)
['first', 'second', 'third']
>>> print(len(thing))
3
```

When you do not specify a delimiter, multiple spaces are treated like one delimiter.

You can specify what delimiter character to use in the splitting.

Tuples Are Like Lists

Tuples are another kind of sequence that functions much like a list - they have elements which are indexed starting at 0

```
>>> x = ('Glenn', 'Sally', 'Joseph')
>>> print(x[2])
Joseph
>>> y = ( 1, 9, 2 )
>>> print(y)
(1, 9, 2)
>>> print(max(y))
9
```

but... Tuples are “immutable”

Unlike a list, once you create a tuple, you cannot alter its contents - similar to a string

```
>>> x = [9, 8, 7]
>>> x[2] = 6
>>> print(x)
>>>[9, 8, 6]
>>>
```

```
>>> y = 'ABC'
>>> y[2] = 'D'
Traceback:'str' object
does
not support item
Assignment
>>>
```

```
>>> z = (5, 4, 3)
>>> z[2] = 0
Traceback:'tuple' object
does
not support item
Assignment
>>>
```

Things not to do With Tuples

```
>>> x = (3, 2, 1)
>>> x.sort()
Traceback:
AttributeError: 'tuple' object has no attribute 'sort'
>>> x.append(5)
Traceback:
AttributeError: 'tuple' object has no attribute 'append'
>>> x.reverse()
Traceback:
AttributeError: 'tuple' object has no attribute 'reverse'
>>>
```

Tuples are More Efficient

- Since Python does not have to build tuple structures to be modifiable, they are simpler and more efficient in terms of memory use and performance than lists.
- When we are making “temporary variables” we prefer tuples over lists.

Tuples and Assignment

- We can also put a tuple on the left-hand side of an assignment statement
- We can even omit the parentheses

```
>>> (x, y) = (4, 'fred')
>>> print(y)
fred
>>> (a, b) = (99, 98)
>>> print(a)
99
```

Tuples are Comparable

The comparison operators work with tuples and other sequences. If the first item is equal, Python goes on to the next element, and so on, until it finds elements that differ.

```
>>> (0, 1, 2) < (5, 1, 2)
True
>>> (0, 1, 2000000) < (0, 3, 4)
True
>>> ( 'Jones', 'Sally' ) < ('Jones', 'Sam')
True
>>> ( 'Jones', 'Sally') > ('Adams', 'Sam')
True
```

Set

- Lists and tuples store values in a sequence.
- Sets are another data type that also store values.
- Sets are a mutable collection of distinct (unique) immutable values that are unordered.
- The major difference is that sets, unlike lists or tuples, cannot have multiple occurrences of the same element and store unordered values.

Source: https://www.datacamp.com/tutorial/sets-in-python

Advantage of Set

- Because sets cannot have multiple occurrences of the same element,
- Set can efficiently remove duplicate values from a list or tuple.
- Set can perform unions and intersections.

Source: https://www.datacamp.com/tutorial/sets-in-python

Create a Set

You can initialize an empty set by using set().

```
>>> emptySet = set()
```

To initialize a set with values, you can pass in a list to set().

```
>>> dataScientist = set(['Python', 'R', 'SQL', 'Git', 'Tableau', 'SAS’])

>>> dataEngineer = set(['Python', 'Java', 'Scala', 'Git', 'SQL',
'Hadoop'])
```

Source: https://www.datacamp.com/tutorial/sets-in-python

Create a Set

- Sets containing values can also be initialized by using curly braces.

```
>>> dataScientist = {'Python', 'R', 'SQL', 'Git', 'Tableau', 'SAS’}
```

- curly braces {} can only be used to initialize a set containing values.

- using curly braces without values is one of the ways to initialize a dictionary and not a set.

```
>>> emptySet = set()
```

```
>>> emptyDict = dict()
>>> emptyDict = {}
```

Source: https://www.datacamp.com/tutorial/sets-in-python

Add Values from Sets

- Initialize set with values.

```
>>> graphicDesigner = {'InDesign', 'Photoshop', 'Acrobat', 'Premiere',
'Bridge’}
```

- You can use the method add to add a value to a set.

```
>>> graphicDesigner.add('Illustrator')
```

Source: https://www.datacamp.com/tutorial/sets-in-python

Remove Values from Sets

- Option 1: remove method removes a value from a set.

```
>>> graphicDesigner.remove('Illustrator’)
```

- Option 2: discard method removes a value from a set. There is no error message even though the value is not in the set.

```
>>> graphicDesigner.discard('Premiere’)
```

- Option 3: pop method removes and returns an arbitrary value from a set.

```
>>> graphicDesigner.pop()
```

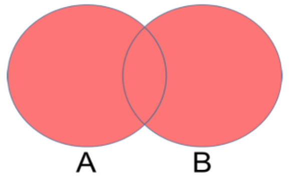
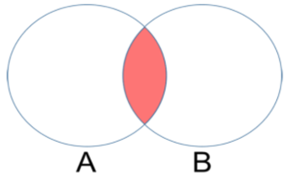
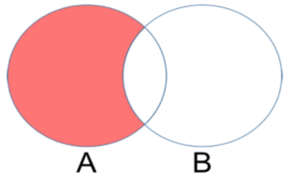
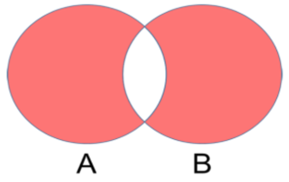
Source: https://www.datacamp.com/tutorial/sets-in-python

Remove Duplicates from a List

```
>>> list(set([1, 2, 3, 1, 7]))
```

Source: https://www.datacamp.com/tutorial/sets-in-python

Set Operations

Set Operation	Venn Diagram	Interpretation
Union		$A \cup B$, is the set of all values that are a member of A, or B, or both.
Intersection		$A \cap B$, is the set of all values that are members of both A and B.
Difference		$A \setminus B$, is the set of all values of A that are not members of B
Symmetric Difference		$A \triangle B$, is the set of all values which are in one of the sets, but not both.

Source: https://www.datacamp.com/tutorial/sets-in-python

Set Operations

```
dataScientist = set(['Python', 'R', 'SQL', 'Git', 'Tableau', 'SAS'])

dataEngineer = set(['Python', 'Java', 'Scala', 'Git', 'SQL', 'Hadoop’])

dataScientist.union(dataEngineer)

dataScientist.intersection(dataEngineer)

dataScientist.difference(dataEngineer)

dataScientist.symmetric_difference(dataEngineer)
```

Source: https://www.datacamp.com/tutorial/sets-in-python

List & Dictionary

- List

  - A linear collection of values that stay in order

- Dictionary

  - A “bag” of values, each with its own label

Dictionaries

- Dictionaries are Python’s most powerful data collection.
- Dictionaries allow us to do fast database-like operations in Python.

Dictionaries

- Lists index their entries based on the position in the list
- Dictionaries are like bags - no order
- So we index the things we put in the dictionary with a “lookup tag”

```
>>> purse = dict()
>>> purse['money'] = 12
>>> purse['candy'] = 3
>>> purse['tissues'] = 75
>>> print(purse)
{'money': 12, 'tissues': 75, 'candy': 3}
>>> print(purse['candy'])
3
>>> purse['candy'] = purse['candy'] + 2
>>> print(purse)
{'money': 12, 'tissues': 75, 'candy': 5}
```

Comparing Lists and Dictionaries

Dictionaries are like lists except that they use keys instead of numbers to look up values

List

```
>>> lst = list()
>>> lst.append(21)
>>> lst.append(183)
>>> print(lst)
[21, 183]
>>> lst[0] = 23
>>> print(lst)
[23, 183]
```

Dictionary

```
>>> ddd = dict()
>>> ddd['age'] = 21
>>> ddd['course'] = 182
>>> print(ddd)
{'course': 182, 'age': 21}
>>> ddd['age'] = 23
>>> print(ddd)
{'course': 182, 'age': 23}
```

Source: Python for Everybody, Charles Severance, University of Michigan School of Information

Dictionary Literals (Constants)

- Dictionary literals use curly braces and have a list of key : value pairs.

- You can make an empty dictionary using empty curly braces.

```
>>> jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
>>> print(jjj)
{'jan': 100, 'chuck': 1, 'fred': 42}
>>> ooo = { }
>>> print(ooo)
{}
```

Many Counters with a Dictionary

One common use of dictionaries is counting how often we “see” something

```
>>> ccc = dict()
>>> ccc['csev'] = 1
>>> ccc['cwen'] = 1
>>> print(ccc)
{'csev': 1, 'cwen': 1}
>>> ccc['cwen'] = ccc['cwen'] + 1
>>> print(ccc)
{'csev': 1, 'cwen': 2}
```

Dictionary Tracebacks

- It is an error to reference a key which is not in the dictionary
- We can use the in operator to see if a key is in the dictionary

```
>>> ccc = dict()
>>> print(ccc['csev'])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'csev'
>>> 'csev' in ccc
False
```

Retrieving Lists of Keys and Values

You can get a list of keys, values, or items (both) from a dictionary

```
>>> jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
>>> print(list(jjj))
['jan', 'chuck', 'fred']
>>> print(jjj.keys())
['jan', 'chuck', 'fred']
>>> print(jjj.values())
[100, 1, 42]
>>> print(jjj.items())
[('jan', 100), ('chuck', 1), ('fred', 42)]
```

Tuples and Dictionaries

The items() method in dictionaries returns a list of (key, value) tuples

```
>>> d = dict()
>>> d['csev'] = 2
>>> d['cwen'] = 4
>>> for (k,v) in d.items():
...     print(k, v)
...
csev 2
cwen 4
>>> tups = d.items()
>>> print(tups)
dict_items([('csev', 2), ('cwen', 4)])
```

Summary: Data structure

- List
- Set
- Tuples
- Dictionary

Acknowledgements / Contributions