notebook_4_getting_started_with_sequences

March 21, 2018

1 Getting Started with Sequential Data in Python

1.1 Objectives

1. Understand operations on sequences
2. Access data from a sequences using an index
3. Access a portion of a sequence using slicing
4. Understand the list comprehension syntax
5. Demonstrate list processing with comprehensions
6. Use list comprehensions in probability simulations

1.2 Three data types

- List
- String
- Tuple

```
In [7]: L = [1,2,3]
        type(L)

Out[7]: list

In [8]: s = "Bob"
        type(s)

Out[8]: str

In [9]: tup = (1,2,3)
        type(tup)

Out[9]: tuple
```

1.2.1 Basic Sequence Operations

Operation	Purpose
+	concatenate
	replicate

Operation	Purpose
s[i]	index
s[i:j]	slice
len(s)	length
s in t	membership
s not in t	membership

1.3 More about indexing

```
In [1]: s = "Hello Bob"
        s[3]

Out[1]: 'l'

In [2]: s[-2]

Out[2]: 'o'

In [3]: L = ['A', 'B', 'C', 'D', 'F']
        L[0]

Out[3]: 'A'

In [4]: L[-4]

Out[4]: 'B'
```

1.4 Slicing

```
In [24]: s[1:7]

Out[24]: 'ello B'

In [25]: s[:4]

Out[25]: 'Hell'

In [26]: s[2:]

Out[26]: 'llo Bob'

In [27]: s[:]

Out[27]: 'Hello Bob'

In [28]: s[1::2]

Out[28]: 'el o'
```

1.5 Slicing works for all sequences

```
In [29]: L[1:7]

Out[29]: ['B', 'C', 'D', 'F']

In [30]: tup[1:]

Out[30]: (2, 3)
```

1.6 Arithmetic

```
In [1]: "123" + "abc"

Out[1]: '123abc'

In [2]: [1,2,3] + ["a","b","c"]

Out[2]: [1, 2, 3, 'a', 'b', 'c']

In [3]: 3*[1,2,3]

Out[3]: [1, 2, 3, 1, 2, 3, 1, 2, 3]

In [4]: 3*"Wow" + 4*"!"

Out[4]: 'WowWowWow!!!!'

In [5]: 2*('a', 'b') + ('c',)

Out[5]: ('a', 'b', 'a', 'b', 'c')
```

1.7 Boolean expressions

```
In [6]: 1 in [1,2,3]

Out[6]: True

In [7]: 5 in [1,2,3]

Out[7]: False

In [8]: "a" not in "Todd"

Out[8]: True

In [9]: "a" in ["a", "b", "c"]

Out[9]: True

In [10]: "a" in ["abc", "def"]

Out[10]: False

In [11]: "todd" == "Todd"

Out[11]: False
```

1.8 Making a range of numbers

- `range` returns a sequence of numbers
- Lazy, converted to a list
  - for small ranges

```
In [13]: range(5)

Out[13]: range(0, 5)

In [14]: list(range(5))

Out[14]: [0, 1, 2, 3, 4]
```

1.9 One argument

- Starts at 0
  - aligned with Python indexes
- Up to, but not including, argument
  - `range(n)` returns `n` elements
  - Useful for repetition

```
In [27]: list(range(10))

Out[27]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

1.10 Two Arguments

- Starts at first argument
- Goes up to, but not including, second argument
  - Like slicing

```
In [15]: list(range(2, 10))

Out[15]: [2, 3, 4, 5, 6, 7, 8, 9]
```

1.11 Three Arguments

- First two as before
- Third argument is step size

```
In [16]: list(range(1,5,2))

Out[16]: [1, 3]

In [17]: list(range(10,2,-1))

Out[17]: [10, 9, 8, 7, 6, 5, 4, 3]
```

1.12 Other list processing functions

1.12.1 sum and max

```
In [23]: sum([1,2,3])
```

```
Out[23]: 6
```

```
In [24]: max([1,2,3])
```

```
Out[24]: 3
```

1.13 all and any

```
In [25]: all([True, False, False]) # True if all entries are True
```

```
Out[25]: False
```

```
In [26]: any([True, False, False]) # True if any entries are True
```

```
Out[26]: True
```

1.13.1 `sorted` - making a new sorted sequence

```
In [3]: sorted([1,3,2,5,4]) # returns a new sorted list
```

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

1.14 Combining lists with `zip`

```
In [30]: zip([1,2,3], ["a", "b", "c"]) # zip is lazy
```

```
Out[30]: <zip at 0x10410f748>
```

```
In [1]: list(zip([1,2,3], ["a", "b", "c"])) # Use list to complete
```

```
Out[1]: [(1, 'a'), (2, 'b'), (3, 'c')]
```

1.14.1 Exercise 1

Write a function named `add_elements` that will add the corresponding elements of two lists.
Example `add_elements([1,2,3], [1,2,3]) == [2, 4, 6]`
HINT: `zip` and `sum` will be helpful here!

```
In [ ]:
```

1.14.2 Exercise 2

Write a function named `largest_three` that will return the three largest elements of a list.
Example largest_three(range(5)) == [4, 3, 2]
Hint `sorted` and slicing should do the trick!

```
In [ ]:
```