Leiden University Medical Center

Python Programming

Data Types

Mihai Lefter

Outline

Lists

Mutable sequences of values.

IPython

```
In [1]: l = [2, 5, 2, 3, 7]
In [2]: type(l)
Out[2]: list
```

Lists can be heterogeneous, but we typically don't use that.

IPython

```
In [3]: a = 'spezi'
In [4]: [3, 'abc', 1.3e20, [a, a, 2]]
Out[4]: [3, 'abc', 1.3e+20, ['spezi', 'spezi', 2]]
```

Tuples

Immutable sequences of values.

IPython

```
In [5]:  t = 'white', 77, 1.5
In [6]:  type(t)
Out[6]:  tuple

In [7]:  color, width, scale = t
In [8]:  width
Out[8]:  77
```

Strings

Immutable sequences of characters.

IPython

```
In [9]:  'a string can be written in single quotes'
Out[9]:  'a string can be written in single quotes'
```

Strings can also be written with double quotes, or over multiple lines with triple-quotes.

IPython

```
In [10]: "this makes it easier to use the ' character"
Out[10]: "this makes it easier to use the ' character"

In [11]: """A multiline string.
         You see? I continued after a blank line."""
Out[11]: 'A multiline string.\n\nYou see?  I continued after a blank line.'
```

Strings

A common operation is formatting strings using argument substitutions.

IPython

```
In [12]: '{} times {} equals {:.2f}'.format('pi', 2, 6.283185307179586)
Out[12]: 'pi times 2 equals 6.28'
```

Accessing arguments by position or name is more readable.

IPython

```
In [13]: '{1} times {0} equals {2:.2f}'.format('pi', 2, 6.283185307179586)
Out[13]: '2 times pi equals 6.28'

In [14]: '{number} times {amount} equals {result:.2f}'.format(number='pi',
             ↪    amount=2, result=6.283185307179586)
Out[14]: 'pi times 2 equals 6.28'
```

Common sequence operations

All sequence types support: concatenation, membership/substring tests, indexing, and slicing.

IPython

```
In [15]: [1, 2, 3] + [4, 5, 6]
Out[15]: [1, 2, 3, 4, 5, 6]

In [16]: 'bier' in 'we drinken bier vanaf half 5'
Out[16]: True

In [17]: 'abcdefghijkl'[5]
Out[17]: 'f'
```

Slicing

Slice `s` from `i` to `j` with `s[i:j]`.

IPython

```
In [18]: 'abcdefghijkl'[4:8]
Out[18]: 'efgh'

In [19]: 'abcdefghijkl'[:3]
Out[19]: 'abc'
```

We can also define the step `k` with `s[i:j:k]`.

IPython

```
In [20]: 'abcdefghijkl'[7:3:-1]
Out[20]: 'hgfe'
```

Several helpful builtins

IPython

```
In [21]: len('attacgataggcatccgt')
Out[21]: 18

In [22]: max([17, 86, 34, 51])
Out[22]: 86

In [23]: sum([17, 86, 34, 51])
Out[23]: 188

In [24]: ('atg', 22, True, 'atg').count('atg')
Out[24]: 2
```

More with lists

We can replace, add, remove, reverse and sort items in-place.

IPython

```
In [25]: l = [1, 2, 3, 4]
In [26]: l[3] = 7
In [27]: l.append(1)
In [28]: l[1:3] = [3, 2]
In [29]: l.sort()
In [30]: l.reverse()
In [31]: l
Out[31]: [7, 3, 2, 1, 1]
```

Additional useful built-ins

IPython

```
In [32]: list('abcdefghijk')
Out[32]: ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k']

In [33]: range(5, 16)  # In python 2: [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
Out[33]: range(5, 16)

In [34]: list(range(5, 16))
Out[34]: [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]

In [35]: zip(['red', 'white', 'blue'], range(3))
Out[35]: <zip at 0x7f3565860108>

In [36]: list(zip(['red', 'white', 'blue'], range(3)))
Out[36]: [('red', 0), ('white', 1), ('blue', 2)]
```

Dictionaries map hashable values to arbitrary objects

IPython

```
In [37]: d = {'a': 27, 'b': 18, 'c': 12}
In [38]: type(d)
Out[38]: dict

In [39]: d['e'] = 17
In [40]: 'e' in d
Out[40]: True

In [41]: d.update({'a': 18, 'f': 2})
In [42]: d
Out[42]: 'a':  18, 'b':  18, 'c':  12, 'e':  17, 'f':  2
```

Accessing dictionary content

IPython

```
In [43]: d['b']
Out[43]: 18

In [44]: d.keys()
Out[44]: dict_keys(['e', 'c', 'a', 'b', 'f'])

In [45]: list(d.keys())
Out[45]: ['a', 'c', 'b', 'e', 'f']

In [46]: list(d.values())
Out[46]: [18, 12, 18, 17, 2]

In [47]: list(d.items())
Out[47]: [('a', 18), ('c', 12), ('b', 18), ('e', 17), ('f', 2)]
```

Sets

Mutable unordered collections of hashable values without duplication

IPython

```
In [48]: x = {12, 28, 21, 17}
In [49]: type(x)
Out[49]: set

In [50]: x.add(12)
In [51]: a
Out[51]: 12, 17, 21, 28

In [52]: x.discard(21)
In [53]: x
Out[53]: 12, 17, 28
```

Sets

Mutable unordered collections of hashable values without duplication

IPython

```
In [54]: x[0]

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-62-2f755f117ac9> in <module>()
----> 1 x[0]

TypeError: 'set' object does not support indexing
```

Operations

We can test for membership and apply many common set operations such as union and intersect.

IPython

```
In [55]: 17 in {12, 28, 21, 17}
Out[55]: True

In [56]: {12, 28, 21, 17} | {12, 18, 11}
Out[56]: 11, 12, 17, 18, 21, 28

In [57]: {12, 28, 21, 17} & {12, 18, 11}
Out[57]: 12
```

Operations

Difference

IPython

```
In [58]: s1 = {12, 28, 21, 17}

In [59]: s2 = {28, 32, 71, 12}

In [60]: s1.difference(s2)
Out[60]: 17, 21
```

Booleans

The two boolean values are written `False` and `True`.

IPython

```
In [61]: True or False
Out[61]: True

In [62]: True and False
Out[62]: False

In [63]: not False
Out[63]: True
```

Comparisons

Comparisons can be done on all objects and return a boolean value.

IPython

```
In [64]: 22 * 3 > 66
Out[64]: False
```

We have two equivalence relations: value equality (==) and object identity (is).

IPython

```
In [65]: a, b = [1, 2, 3], [1, 2, 3]
In [66]: a == b
Out[66]: True

In [67]: a is b
Out[67]: False
```

Hands on!

1. Make a list `l1` with 10 integer elements.
   a What is the sum of all the items in the `l1` list.
   b Make a new list `l2` from `l1` that does not include the 0th, 4th, and 5th elements.
   c Sum only the elements from `l1` which are between the 2nd and the 6th elements.
2. Food:
   a. Create a dictionary for food products called `prices` and put some values in it, e.g., "apples": 2, "oranges": 1.5, "pears": 3, ...
   b. Create a corresponding dictionary called "stocks" and put the stock values in it, e.g., "apples": 0, "oranges": 1, "pears": 10, ...
   c. Add another entry in the `prices` dictionary with key 'bananas' and value 13.
   d. Add another entry in the `stocks` dictionary with key 'bananas' and value 11.
   e. What is the total money value for the "bananas" (stock $\times$ price)?
   f. How many products are in the `stocks` dictionary?
   g. Are the number of products in the `stocks` and `prices` dictionaries equal?
   h. Are there the same products in the `stocks` and `prices` dictionaries?
   i. What is the most expensive value in the `prices` dictionary?

Leiden University Medical Center

Acknowledgements

Martijn Vermaat
Jeroen Laros
Jonathan Vis