Leiden University Medical Center

Python Programming

Data Types

Mark Santcroos

Lists

Mutable sequences of values.

```
>>> l = [2, 5, 2, 3, 7]
>>> type(l)
<class 'list'>
```

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

```
>>> a = 'xyz'
>>> [3, 'abc', 1.3e20, [a, a, 2]]
[3, 'abc', 1.3e+20, ['xyz', 'xyz', 2]]
```

Tuples

Immutable sequences of values.

```
>>> t = 'white', 77, 1.5
>>> type(t)
<class 'tuple'>

>>> color, width, scale = t
>>> width
77
```

Strings

Immutable sequences of characters.

```
>>> 'a string can be written in single quotes'
'a string can be written in single quotes'
```

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

```
>>> "this makes it easier to use the ' character"
"this makes it easier to use the ' character"
```

```
>>> """A multiline string.
... You see? I continued after a blank line."""
'A multiline string.\nYou see? I continued after a blank line.'
```

Strings

But not mix them!

```
>>> 'a string can not be written with mixed quotes"
  File "<stdin>", line 1
    'a string can not be written with mixed quotes"
                                                   ^
SyntaxError: EOL while scanning string literal
```

Strings

A common operation is formatting strings using argument substitutions.

```
>>> '{} times {} equals {:.2f}'\
...     .format('pi', 2, 6.283185307179586)
'pi times 2 equals 6.28'
```

Accessing arguments by position or name is more readable.

```
>>> '{1} times {0} equals {2:.2f}'\
...     .format('pi', 2, 6.283185307179586)
'2 times pi equals 6.28'

>>> '{number} times {amount} equals {result:.2f}'\
...     .format(number='pi', amount=2, result=6.283185307179586)
'pi times 2 equals 6.28'
```

Common sequence operations

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

```
>>> [1, 2, 3] + [4, 5, 6]
[1, 2, 3, 4, 5, 6]

>>> 'hay' in 'haystack'
True

>>> 'needle' in 'haystack'
False

>>> 'abcdefghijkl'[3]
'd'
```

Slicing

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

```
>>> 'abcdefghijkl'[4:8]
'efgh'
>>> 'abcdefghijkl'[:3]
'abc'
```

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

```
>>> 'abcdefghijkl'[7:3:-1]
'hgfe'
```

Several helpful builtins

```
>>> len('attacgataggcatccgt')
18

>>> max([17, 86, 34, 51])
86

>>> sum([17, 86, 34, 51])
188

>>> ('atg', 22, True, 'atg').count('atg')
2
```

More with lists

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

```
>>> l = [1, 2, 3, 4]
>>> l[3] = 7
>>> l.append(1)
>>> l[1:3] = [3, 2]
>>> l.sort()
>>> l.reverse()
>>> l
[7, 3, 2, 1, 1]
```

Additional useful built-ins

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

>>> range(5, 16)
range(5, 16)

>>> list(range(5, 16))
[5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]

>>> zip(['red', 'white', 'blue'], range(3))
<zip object at 0x109d9d380>

>>> list(zip(['red', 'white', 'blue'], range(3)))
[('red', 0), ('white', 1), ('blue', 2)]
```

Dictionaries

Unordered map of hashable values to arbitrary objects

```
>>> d = {'a': 27, 'b': 18, 'c': 12}
>>> type(d)
<class 'dict'>

>>> d['e'] = 17
>>> 'e' in d
True

>>> d.update({'a': 18, 'f': 2})
>>> d
{'a': 18, 'b': 18, 'c': 12, 'e': 17, 'f': 2}
```

Accessing dictionary content

```
>>> d['b']
18

>>> d.keys()
dict_keys(['a', 'b', 'c', 'e', 'f'])

>>> list(d.keys())
['a', 'b', 'c', 'e', 'f']

>>> list(d.values())
[18, 18, 12, 17, 2]

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

Sets

Mutable unordered collections of hashable values without duplication

```
>>> x = {12, 28, 21, 17}
>>> type(x)
<class 'set'>

>>> x.add(12)
>>> x
{17, 21, 12, 28}

>>> x.discard(21)
>>> x
{17, 12, 28}
```

Sets are not indexed

Sets are unordered collections, and therefore without index.

```
>>> x[0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'set' object is not subscriptable
```

In contrast to lists ...

```
>>> list(x)[0]
17
```

Operations

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

```
>>> 17 in {12, 28, 21, 17}
True

>>> {12, 28, 21, 17} | {12, 18, 11}
{17, 18, 21, 11, 12, 28}

>>> {12, 28, 21, 17} & {12, 18, 11}
{12}
```

Operations

Difference

```
>>> s1 = {12, 28, 21, 17}

>>> s2 = {28, 32, 71, 12}

>>> s1.difference(s2)
{17, 21}
```

Booleans

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

```
>>> True or False
True

>>> True and False
False

>>> not False
True
```

Comparisons

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

```
>>> 1 < 2
True

>>> 1 == 2
False

>>> "Left" == "Right"
False

>>> "Right" == "Right"
True
```

Value vs object

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

```
>>> a, b = [1, 2, 3], [1, 2, 3]
>>> a == b
True

>>> a is b
False

>>> a = 0
>>> b = 0
>>> a is b
True
```

Changing the type of a value

Sometimes you might want to combine values of different types.

```
>>> x = 1
>>> name = 'John'
>>> name + x
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```

Combining different types

```
>>> x = 1
>>> name = 'John'
>>> name + str(x)
'John1'
```

And further ...

```
>>> x = 1
>>> x
1
>>> str(x)
'1'
>>> int(str(x))
1
```

Hands on!

1. Make a list `list1` with 10 integer elements.
   a. What is the sum of all the items in the `list1` list.
   b. Make a list `list2` from `list1` that does not include the 0th, 4th, and 5th elements.
   c. Sum only the elements from `list1` 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 `stock` 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

Mihai Lefter
Martijn Vermaat
Jeroen Laros
Jonathan Vis