



PyQB

Monga

Composite
objects

Tuples and lists

Dictionaries

Sets

Comprehensions

Programming in Python¹

Mattia Monga

Dip. di Informatica
Università degli Studi di Milano, Italia

`mattia.monga@unimi.it`

Academic year 2021/22, II semester



PyQB

Monga

Composite
objects

Tuples and lists

Dictionaries

Sets

Comprehensions

Lecture VII: Composite objects



Simple and composite objects

- `ints` `floats` `bools` are simple objects: they have no “parts”
- Strings are an example of composite objects since it is possible to consider also the characters: a `str` is a sequence of single characters; an important (simplifying) property: they are `immutable`
- Generic `immutable` sequences (with elements of any type) are called `tuples` (`tuple`): `(1, 2, 'foo')` `(1,)`
- Generic `mutable` sequences (with elements of any type) are called `lists` (`list`): `[1, 2, 'foo']` `[1]`
`[1,2].append(3)`

PyQB

Monga

Composite
objects

Tuples and lists

Dictionaries

Sets

Comprehensions



Mutability

Immutable objects are simpler to use:

```
x = (1, 2, 3)
y = x
```

```
x = (10, 20, 30) # x refers to a new object, since the
                 ↪ old cannot be changed
```

```
print(x, y)
```

Mutable ones require some caution:

```
x = [1, 2, 3]
y = x
```

```
x[0] = 10 # both x and y refer to a changed object
print(x, y)
```

```
x = [100, 200, 300]
print(x, y)
```

```
z = x[:] # a copy not the same object
```

PyQB

Monga

Composite
objects

Tuples and lists

Dictionaries

Sets

Comprehensions



- Write a function `middle(L)` which takes a list L as its argument, and returns the item in the middle position of L . (In order that the middle is well-defined, you should assume that L has odd length.) For example, calling `middle([8, 0, 100, 12, 1])` should return 100, since it is positioned exactly in the middle of the list.
(`assert` is a useful tool to check assumptions — known as `preconditions` — are indeed true)
- Define a function `prod(L)` which returns the product of the elements in a list L .
(for type hinting it is sometimes useful the catch-all type `Any` (`from typing import Any`))



Dictionaries

A composite type `dict` that implements a `mapping` between immutable `keys` and `values`.

```
d = {'key': 'foo', 3: 'bar'}
```

```
print(d['key']) # 'foo'  
print(d[3])     # 'bar'  
print(d[2])     # error!
```

Notation is similar to lists/tuples, but `dicts` are not sequences indexed by numbers, you must use only the existing keys (`d.keys()`).

```
if x in d.keys():  
    print(d[x])
```

A sequence of values can be obtained with `d.values`. A sequence of 2-tuples (key, value) with `d.items()`.

PyQB

Monga

Composite
objects

Tuples and lists

Dictionaries

Sets

Comprehensions



PyQB

Monga

Composite
objects

Tuples and lists

Dictionaries

Sets

Comprehensions

A `set` is a composite object with no duplicate (non mutable) elements. Common set operations are possible.

- Set literals: `{1,2,3}` `set()`
- `{1,2,3}.union({3,5,6})`
`{1,2,3}.intersection({3,5,6})`



Comprehensions

Comprehensions are a concise way to create lists, sets, maps... It resembles the mathematical notation used for sets

$$A = \{a^2 | a \in \mathbb{N}\}.$$

```
squares = [x**2 for x in range(10)]
```

equivalent to:

```
squares = []  
for x in range(10):  
    squares.append(x**2)
```

filtering is possible

```
odds = [x for x in range(100) if x % 2 != 0]
```

with a set

```
s = {x for x in range(50+1) if x % 5 == 0}
```

with a dict

```
d = {x: x**2 for x in range(10)}
```

PyQB

Monga

Composite
objects

Tuples and lists

Dictionaries

Sets

Comprehensions