Programming in Python[1]

Mattia Monga

Dip. di Informatica
Università degli Studi di Milano, Italia
mattia.monga@unimi.it

Academic year 2023/24, I semester

Lecture VII: Composite objects

Simple and composite objects

- `int`s `float`s `bool`s 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)`

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.copy() # a copy not the same object
```

Exercises

- Write a function `middle(L: list[int])` 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: list[int])` which returns the product of the elements in a list *L*.

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 `dict`s 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()`.