Sorting and Object Oriented Programming

Basic Programming in Python

Sebastian Höffner   Aline Vilks

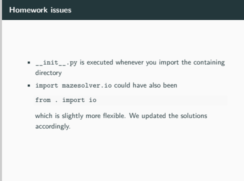
Wed, 17 May 2017

Homework issues

- `__init__.py` is executed whenever you import the containing directory
- `import mazesolver.io` could have also been

  ```
  from . import io
  ```

  which is slightly more flexible. We updated the solutions accordingly.



Homework issues

- `__init__.py` is executed whenever you import the containing directory
- `import mazesolver.io` could have also been

```
from . import io
```

which is slightly more flexible. We updated the solutions accordingly.

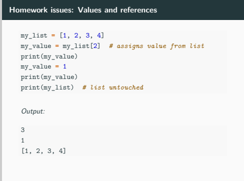
`from . import io` means: “From the directory the current module is in (`__init__.py`’s directory), import the module `io` (here `io.py`)”

Homework issues: Values and references

```python
my_list = [1, 2, 3, 4]
my_value = my_list[2]  # assigns value from list
print(my_value)
my_value = 1
print(my_value)
print(my_list)  # list untouched
```

Output:

```
3
1
[1, 2, 3, 4]
```



Homework issues: Values and references

```
my_list = [1, 2, 3, 4]
my_value = my_list[2]  # assigns value from list
print(my_value)
my_value = 1
print(my_value)
print(my_list)  # list untouched
```

Output:

```
3
1
[1, 2, 3, 4]
```

Python always copies values to new variables.

For simple types (int, float, etc.) Python copies values.

For complex types (lists, dictionaries, functions, instances (today)) Python copies the references.

A reference is just a hint to the place where the data is stored.

Homework issues: Values and references

```python
my_list = [1, 2, 3, 4]
my_other_list = my_list  # assigns reference
print(my_other_list)
my_other_list[2] = 1  # changes BOTH lists
print(my_other_list)
print(my_list)
```

Output:

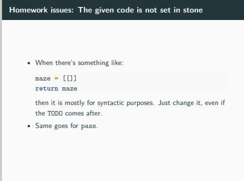
```
[1, 2, 3, 4]
[1, 2, 1, 4]
[1, 2, 1, 4]
```

Homework issues: The given code is not set in stone

- When there's something like:

  ```
  maze = [[]]
  return maze
  ```

  then it is mostly for syntactic purposes. Just change it, even if the `TODO` comes after.
- Same goes for `pass`.



Homework issues: The given code is not set in stone

- When there's something like:

  ```
  maze = [[]]
  return maze
  ```

  then it is mostly for syntactic purposes. Just change it, even if the TODO comes after.
- Same goes for pass.

pass is a NOOP, a "**No Op**eration" statement. Its only purpose is to make unfinished code syntactically correct.

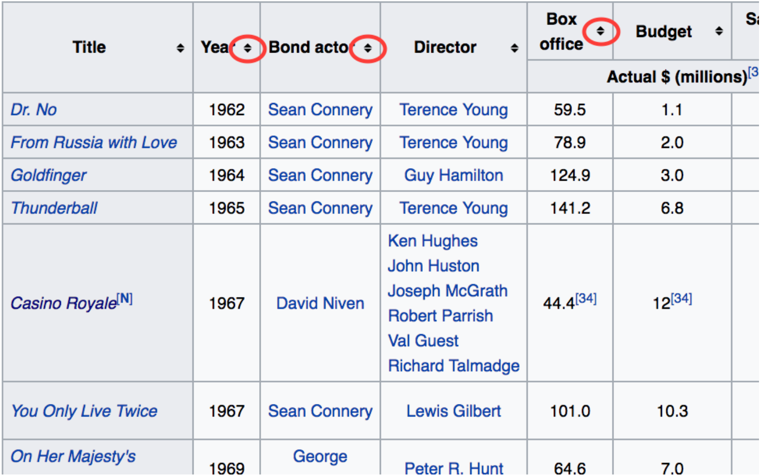
Sorting

Why do we need to sort data?

- Searching is easier.
- Data is easier to understand.
- Rankings can be performed.
- ...

In many applications, sorting is just a preprocessing step to allow further data processing.

Sort example

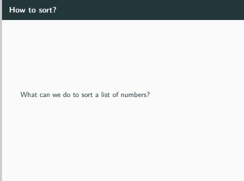


Title ⬍	Year ⬍	Bond actor ⬍	Director ⬍	Box office ⬍	Budget ⬍	Sa
				Actual $ (millions)[3		
Dr. No	1962	Sean Connery	Terence Young	59.5	1.1	
From Russia with Love	1963	Sean Connery	Terence Young	78.9	2.0	
Goldfinger	1964	Sean Connery	Guy Hamilton	124.9	3.0	
Thunderball	1965	Sean Connery	Terence Young	141.2	6.8	
Casino Royale[N]	1967	David Niven	Ken Hughes John Huston Joseph McGrath Robert Parrish Val Guest Richard Talmadge	44.4[34]	12[34]	
You Only Live Twice	1967	Sean Connery	Lewis Gilbert	101.0	10.3	
On Her Majesty's	1969	George	Peter R. Hunt	64.6	7.0	

Figure 1: Wikipedia's List of Bond Movies

How to sort?

What can we do to sort a list of numbers?



How to sort?

What can we do to sort a list of numbers?

The most intuitive way is to search the smallest number and put it in front. Then search the second smallest number and add it to the second position. And so on.

This procedure is called “selection sort”. It is a fun exercise at home, but not really useful in practice.

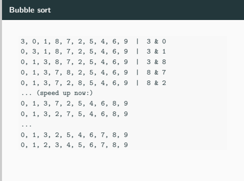
Bubble sort interactive

Bubble sort[1]

[1] https://www.youtube.com/watch?v=lyZQPjUT5B4

Bubble sort

```
3, 0, 1, 8, 7, 2, 5, 4, 6, 9  |  3 & 0
0, 3, 1, 8, 7, 2, 5, 4, 6, 9  |  3 & 1
0, 1, 3, 8, 7, 2, 5, 4, 6, 9  |  3 & 8
0, 1, 3, 7, 8, 2, 5, 4, 6, 9  |  8 & 7
0, 1, 3, 7, 2, 8, 5, 4, 6, 9  |  8 & 2
... (speed up now:)
0, 1, 3, 7, 2, 5, 4, 6, 8, 9
0, 1, 3, 2, 7, 5, 4, 6, 8, 9
...
0, 1, 3, 2, 5, 4, 6, 7, 8, 9
0, 1, 2, 3, 4, 5, 6, 7, 8, 9
```



Bubble sort

```
3, 0, 1, 8, 7, 2, 5, 4, 6, 9  |  3 & 0
0, 3, 1, 8, 7, 2, 5, 4, 6, 9  |  3 & 1
0, 1, 3, 8, 7, 2, 5, 4, 6, 9  |  3 & 8
0, 1, 3, 7, 8, 2, 5, 4, 6, 9  |  8 & 7
0, 1, 3, 7, 2, 8, 5, 4, 6, 9  |  8 & 2
... (speed up now:)
0, 1, 3, 7, 2, 5, 4, 6, 8, 9
0, 1, 3, 2, 7, 5, 4, 6, 8, 9
...
0, 1, 3, 2, 5, 4, 6, 7, 8, 9
0, 1, 2, 3, 4, 5, 6, 7, 8, 9
```

- Bubble sort compares the first two elements.
- If the first is greater than the second, it swaps them.
- Then it compares the (possibly new) second and third elements and swaps if needed.
- This process is done until it went through the list once.
- If at any point a swap was needed, the process is repeated.
- If not, the list is sorted.

Bubble sort

File: `bubblesort.py`

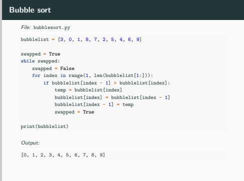
```python
bubblelist = [3, 0, 1, 8, 7, 2, 5, 4, 6, 9]

swapped = True
while swapped:
    swapped = False
    for index in range(1, len(bubblelist[1:])):
        if bubblelist[index - 1] > bubblelist[index]:
            temp = bubblelist[index]
            bubblelist[index] = bubblelist[index - 1]
            bubblelist[index - 1] = temp
            swapped = True

print(bubblelist)
```

Output:

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



Bubble sort

File: bubblesort.py

```
bubblelist = [3, 0, 1, 8, 7, 2, 5, 4, 6, 9]

swapped = True
while swapped:
    swapped = False
    for index in range(1, len(bubblelist[1:])):
        if bubblelist[index - 1] > bubblelist[index]:
            temp = bubblelist[index]
            bubblelist[index] = bubblelist[index - 1]
            bubblelist[index - 1] = temp
            swapped = True

print(bubblelist)
```

Output:

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

While bubble sort is easy to implement and talk about, it is also not really useful in practice.

Instead, quicksort is a very popular choice. If you are interested in it, take a look at its Wikipedia page. For most applications its the only sorting algorithm you need to know about.

https://en.wikipedia.org/wiki/Quicksort

Sorting something else than boring numbers

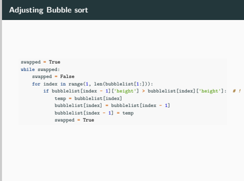
In the fifth homework we modeled persons:

```
person = {
    'name': 'Alecia',
    'age': 37,
    'height': 1.63
}
```

Let's sort some of them!

Adjusting Bubble sort

```
swapped = True
while swapped:
    swapped = False
    for index in range(1, len(bubblelist[1:])):
        if bubblelist[index - 1]['height'] > bubblelist[index]['height']:  # !
            temp = bubblelist[index]
            bubblelist[index] = bubblelist[index - 1]
            bubblelist[index - 1] = temp
            swapped = True
```



Adjusting Bubble sort

```
swapped = True
while swapped:
    swapped = False
    for index in range(1, len(bubblelist[1:])):
        if bubblelist[index - 1]['height'] > bubblelist[index]['height']:  # !
            temp = bubblelist[index]
            bubblelist[index] = bubblelist[index - 1]
            bubblelist[index - 1] = temp
            swapped = True
```

The only thing to adjust is which values to compare. Since our list now contains dictionaries, we can access the value behind the key `height` to sort by height.

Sorting persons

File: personsort.py

```
import peopledb
persons = peopledb.read('persons.data')
print(len(persons), 'Example:', persons[0])
persons = peopledb.bubblesort(persons)
for person in persons[:4]:
    print(person)
```

Output:

```
14 Example: {'name': 'Alecia', 'age': 37, 'height': 1.63}
{'name': 'Susanna', 'age': 15, 'height': 1.46}
{'name': 'Gertrude', 'age': 64, 'height': 1.52}
{'name': 'Bertha', 'age': 45, 'height': 1.59}
{'name': 'Alecia', 'age': 37, 'height': 1.63}
```

Dictionaries vs. Classes and Objects

- Dictionaries are meant for key-value mappings, not for modeling
- There's a much better concept: Classes and objects

Dictionaries vs. Classes and Objects

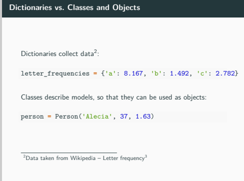
Dictionaries collect data[2]:

```
letter_frequencies = {'a': 8.167, 'b': 1.492, 'c': 2.782}
```

Classes describe models, so that they can be used as objects:

```
person = Person('Alecia', 37, 1.63)
```

[2] Data taken from Wikipedia – Letter frequency[3]



Dictionaries vs. Classes and Objects

Dictionaries collect data[2]:

```
letter_frequencies = {'a': 8.167, 'b': 1.492, 'c': 2.782}
```

Classes describe models, so that they can be used as objects:

```
person = Person('Alecia', 37, 1.63)
```

[2] Data taken from Wikipedia – Letter frequency[3]

Dictionaries can be extended if needed, whenever a new key-value pair makes sense. They should not be used to model many instances, but to collect data with meaningful keys.

Classes describe a concept of which one can instantiate objects (or instances, the terms are used interchangeably). In other words, classes are the **blueprint**s, while objects are the **realization**s.

Modeling a person

Live demo: Let's model a person!

Modeling a person

```python
class Person:

    def __init__(self, name, age, height):
        self.name = name
        self.age = age
        self.height = height

    def introduce(self):
        return 'Hi, I am ' + self.name

person = Person('Alecia', 37, 1.63)
print(person)
print(person.introduce())
```

Output:

```
<__main__.Person object at 0x108d2cf60>
Hi, I am Alecia
```

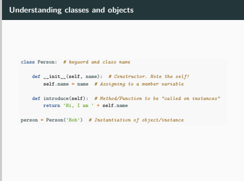
Understanding classes and objects

```python
class Person:  # keyword and class name

    def __init__(self, name):  # Constructor. Note the self!
        self.name = name  # Assigning to a member variable

    def introduce(self):  # Method/Function to be "called on instances"
        return 'Hi, I am ' + self.name

person = Person('Bob')  # Instantiation of object/instance
```



Understanding classes and objects

```
class Person:  # keyword and class name

    def __init__(self, name):  # Constructor. Note the self!
        self.name = name  # Assigning to a member variable

    def introduce(self):  # Method/Function to be "called on instances"
        return 'Hi, I am ' + self.name

person = Person('Bob')  # Instantiation of object/instance
```

Each class starts with the keyword `class` followed by the class name (here `Person`) and introduces a new block, the *class body*.

Classes can have a constructor, usually used to assign member variables. A constructor is always called `def __init__(self):` Notice the `__` again, it's a special Python function. By implementing it, the "default" implementation (which does nothing) is overwritten.

`self` is very special. While the name does not matter, by convention we use `self` as the first argument: it is always a reference to the *calling object*. So if we have a `person = Person('Tom')`, then `person.introduce()` uses `person` as `self` (the first argument).

All functions declared inside the class body are often called *methods* (and I will do that often), but they really are just functions which can be called by instances of the class.

Modeling other objects

```python
class Car:
    def __init__(self, color, speed=1):
        self.color = color
        self.speed = speed
        self.distance = 0

    def drive(self):
        self.distance += self.speed

cars = [Car('indigo'), Car('firebrick', 3)]
for car in cars:
    car.drive()
    print(car.color, car.distance)
```

Output:

```
indigo 1
firebrick 3
```

Ugly print

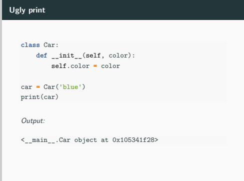
```python
class Car:
    def __init__(self, color):
        self.color = color

car = Car('blue')
print(car)
```

Output:

```
<__main__.Car object at 0x105341f28>
```

└Ugly print



Ugly print

```
class Car:
    def __init__(self, color):
        self.color = color

car = Car('blue')
print(car)
```

Output:

```
<__main__.Car object at 0x105341f28>
```

By default the print output of objects looks like this:

```
<__main__.Car object at 0x108e09358>
```

This is the module name, followed by the class name and the type of what you are printing (an object). The hexadecimal number (`0x108e09358`) is the memory address, so where Python stores the data.

Beautiful print

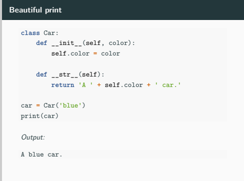
```python
class Car:
    def __init__(self, color):
        self.color = color

    def __str__(self):
        return 'A ' + self.color + ' car.'

car = Car('blue')
print(car)
```

Output:

```
A blue car.
```



Beautiful print

```
class Car:
    def __init__(self, color):
        self.color = color

    def __str__(self):
        return 'A ' + self.color + ' car.'

car = Car('blue')
print(car)
```

Output:

```
A blue car.
```

The `def __str__(self)` method is again special and allows for nicer print outputs (= string representations).

It always only takes one argument (`self`) and **must** return a string.

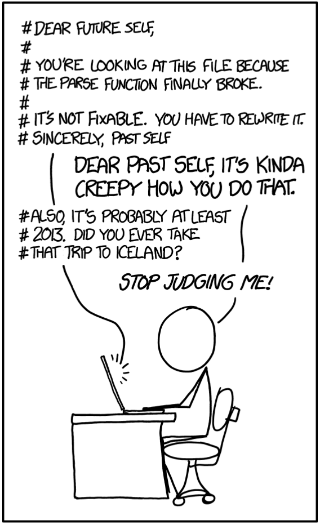
Calling `str(my_object)` causes Python to search for the `__str__` method.

There are many more of these special methods like `__init__` and `__str__`, but for now these should be sufficient.

Your o o seventh homework

- Model a movie. Use a database of Bond movies to create instances. Sort the movies by their release year.
- Revisit the Castle Crashers exercise from sheet 2: Let's model them with proper classes.

The last slide



# DEAR FUTURE SELF,
#
# YOU'RE LOOKING AT THIS FILE BECAUSE
# THE PARSE FUNCTION FINALLY BROKE.
#
# IT'S NOT FIXABLE. YOU HAVE TO REWRITE IT.
# SINCERELY, PAST SELF
DEAR PAST SELF, IT'S KINDA CREEPY HOW YOU DO THAT.
#ALSO, IT'S PROBABLY AT LEAST
# 2013. DID YOU EVER TAKE
#THAT TRIP TO ICELAND?
STOP JUDGING ME!

Figure 2: Maybe I haven't been to Iceland because I am dealing with YOUR crummy code. (Munroe 2014)

References

Munroe, Randall. 2014. “Future Self.” *Xkcd. A Webcomic of Romance, Sarcasm, Math, and Language.*, no. 1421 (September).