Programming

More Python Data Structures

Vassilis Markos, Mediterranean College

Week 06

Contents

1. Last Time's Exercises
2. More Python Data Structures
3. Fun Time!

Last Time’s Exercises

In–class Exercise #001

Using appropriate Python data structures write a Python script that:

- Asks the user to provide student names along their grade at the Programming module.
- Stores the above in an appropriate data structure.
- Asks the user to provide a student name and prints their grade.
- The script should terminate when the user enters an empty student name.

In–class Exercise #002

Using appropriate Python data structures write a Python script that:

- asks the user to provide positive integers (number insertion is terminated when the user inserts a non–positive integer);
- computes the sum of each integer's proper divisors;
- then asks the user repeatedly for numbers and prints on screen the sum of its proper divisors;
- the script should terminate when the user asks for the divisor sum of a non–positive integer.

In–class Exercise #003

Read the following Wikipedia lemma on the Sieve of Eratosthenes:

`https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes`

Then, using appropriate Python data structures, implement the sieve algorithm that computes all prime numbers up to a certain provided positive integer, `n`.

In–class Exercise #004

Since programming is not only about reading simple input from the user and returning a nicely computed output, but also about crafting some more complex projects, how about we build our own game?
In today's materials, under the following link:

`../labs/Programming_Lab_01.pdf`

you can find the first part of a Lab series for this course. Follow the instructions found therein to start working on it!

More Python Data Structures

Unique Entries

What is the functionality of the following script?

```
1  # source/unique_001.py
2
3  def read_ints():
4      numbers = []
5      n = int(input("Please, enter an integer: "))
6      while n > 0:
7          numbers.append(n)
8          n = int(input("Please, enter an integer: "))
9      return numbers
10
11 def get_unique(numbers):
12     unique = []
13     for n in numbers:
14         if n not in unique:
15             unique.append(n)
16     return unique
17
18 if __name__ == "__main__":
19     ns = read_ints()
20     unique = get_unique(ns)
21     print(unique)
```

Unique Entries

As you might have guessed, the previous script reads some user input and finds all unique entries. For instance:

- input: `[4, 5, 7, 1, 2]` $\rightarrow$ `[4, 5, 7, 1, 2]`.
- input: `[4, 5, 7, 4, 5]` $\rightarrow$ `[4, 5, 7]`.
- input: `[0, 0, 0, 0, 0]` $\rightarrow$ `[0]`.
- input: `[7, 1, 3, 8, 7]` $\rightarrow$ `[7, 1, 3, 8]`.

But, is this an efficient way to do so?

Python Sets

Python offers a data structure tailored for such cases: sets.

- Python sets are much like lists, in terms of being collections of items, however **each element can appear at most once.**
- So, **no duplicates** can ever exist in a set.
- This comes at the cost of losing element ordering, in the sense this is allowed in lists.
  - So, if `s` is a python set, we cannot say things like `s[6]` or `s[-1]`, as with lists.
  - Also, **element order is not preserved** in any way, so, in case you need elements to be stored in a certain order, do not use sets!

Python Sets

So, we can improve the above by using a set instead of a list:

```
# source/unique_002.py

def read_ints():
    numbers = set() # Initialise an empty set
    n = int(input("Please, enter an integer: "))
    while n > 0:
        numbers.add(n) # .add() is like .append() for lists
        n = int(input("Please, enter an integer: "))
    return numbers

if __name__ == "__main__":
    ns = read_ints()
    print(ns)
```

What Will This Print?

```
# source/sets_001.py

if __name__ == "__main__":
    a = set([1, 3, 4, 6]) # Initialise a set from a list
    b = { 1, 4, 7, 9 } # Initialise a set directly
    print(a.intersection(b))
    print(a.union(b))
    print(a.difference(b))
    print(a.symmetric_difference(b))
```

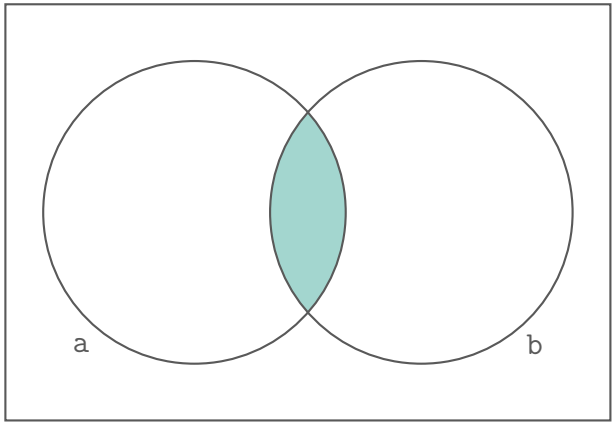
Key Set Operations

Expected output:

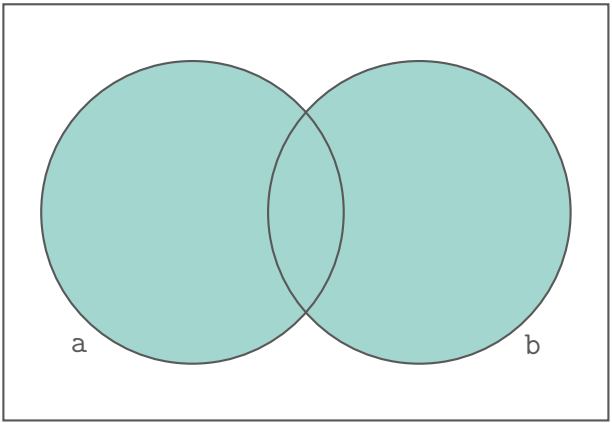
```
{1, 4}
{1, 3, 4, 6, 7, 9}
{3, 6}
{3, 6, 7, 9}
```

- Python sets support all common maths set operations, such as intersection, union, etc.
- All such functions are called from a certain set and **return a new set**.
- There are also `_update()` variants that update the calling set in place.

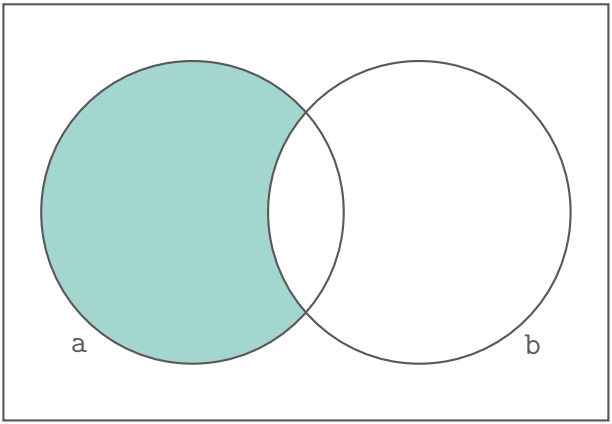
a.intersection(b)



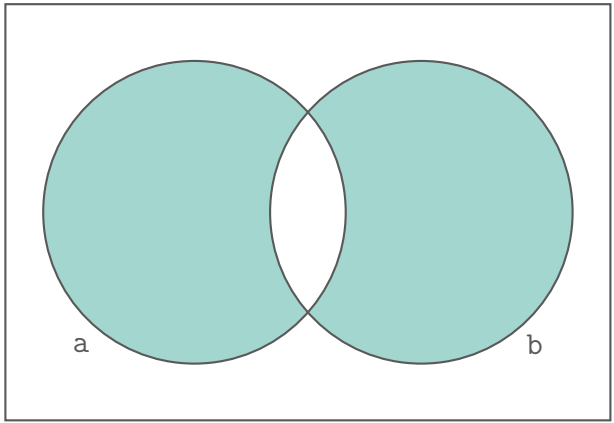
a.union(b)



a.difference(b)



a.symmetric_difference(b)



Python Comprehensions

Python offers some fancy one–liners to create lists, sets and dictionaries:

- List comprehension: `[x for x in range(5)]` creates the list `[0, 1, 2, 3, 4]`.
- Set comprehension: `{x for x in range(5)}` creates the set `{0, 1, 2, 3, 4}`.
- Dictionary comprehension: `{x: x for x in range(5)}` creates the dictionary `{0:0, 1:1, 2:2, 3:3, 4:4}`.

For more: `https://docs.python.org/3/tutorial/datastructures.html#list-comprehensions`

List Slicing

Last time we forgot to mention a nice Python list capability: **list slicing.**

- For instance, to get the elements of a list, `a`, starting from position 5 up to position 8, we can write: `a[5:9]` (the right end is always excluded).
- To get everything from start to position 5 with a step of 2: `a[:6:2]`.
- To get the list in reverse order: `a[::-1]`.

Fun Time!

In–class Exercise #001

In today's materials, under the following link:

`../labs/Programming_Lab_02.pdf`

you can find the second part of a Lab series for this course. Follow the instructions found therein to start working on it!

In–class Exercise #002

Start working on all Labs found in today's materials `homework` directory.
To help me asses those files, you can name them as follows:

`task_xxx.py`

where `xxx` is the number of the task. For instance, task 5 file could be named `task_005.py`.
Submit your work via email at: `v.markos@mc-class.gr`

Homework

- In this week's materials, under the `homework` directory, you can find some Python programming Tasks. Complete as many of them as you can (preferably all).
- This is important, since tasks such as those provided with this lecture's materials will most probably be part of your course assessment portfolio. So, take care to solve as many of those tasks as possible!
- Share your work at: `v.markos@mc-class.gr`

Any Questions?

Do not forget to fill in the questionnaire shown right!

https://forms.gle/dKSrmE1VRVWqxBGZA