Fundamental types and collections

Estimated time for completion: 45 minutes

Overview:

In this lab, you will practice writing code to use fundamental types (numbers, strings, etc.) and collections.

These instructions assume you have **Python 3** installed. You can use any OS you choose (OS X, Windows, or Linux).

Goals:

- Convert numbers to and from strings
- Convert numbers between types (e.g. float to int)
- Compute relationships between dates
- Create and manipulate collections
- Use slicing to efficiently access subsets of strings and lists

Part 1 – Working with numbers

There is no 'starter code' for this lab. Just create a new project or folder to hold your files. Add a file called numbers.py.

You will write some code to query the user for a complex number (these are of the form X+Yj where X & Y floating point or integer numbers). Review the output from this code below (not yet written). The Python data structure is named complex and is already imported.

```
Working with numbers:


What is your favorite *complex* number [x + yj]? 5

Come on, that's not a very imaginative imaginary number, try again.


What is your favorite *complex* number [x + yj]? 7j

Come on, that's not a very imaginative imaginary number, try again.


What is your favorite *complex* number [x + yj]? 3+5j

Oh, that's a good one

Here are some facts about (3+5j).

Length: 5.830951894845301 which is approximately 6.

The real part is 3.0 and the imaginary part is 5.0

The real part is not divisible by 2.
```

You will need following formula (NOT valid Python code) for computing the length of a **complex** number:

Length = sqrt(real_part(z * z_conjugate))

If case you don't remember this from math, don't worry: conjugate and real are methods on complex numbers. For square roots, you will need to import the **math** module.

It may also be helpful to know that you can round floating point numbers by adding .5 to them and then converting them to integers.

Your task: Use the output listed above and hints given just following it to implement this part of the lab.

Part 2 – Working with dates, times, and time spans

Now it's time to work with dates and time spans (represented by the **timedelta** class in Python). You should create a new file called **workwithdates.py**. To get started, import the **datetime** module. Using datetime.date, datetime.date.today, and the fact that you can do math between datetimes and timedeltas to write a program with the output below. For example:

- datetime1 – datetime2 => timedelta
- datetime1+timedelta => datetime

Note: You need to be careful of the case where a person's birthday has already occurred that year. For example, if they were born March 1st, and it's June 7th, you need to adjust for this.

workingwithdates.py output:

```
Working with dates:


Right now it's 2014-04-10

What year were you born? 1973

What month were you born? 5

What day were you born? 1

Ok, your birthday is 5/1/1973

Looks like you are 40 years old

Hope you're looking forward to your birthday. It's in 21 days.
```

Part 3 – Working with collections

Collections play a key role in Python. Let’s look at the three most common ones: lists, dictionaries, and sets. Tuples are also quite common, but we won’t address tuples right now. Begin by adding a new python file to your project named collections.py.

In this section, we will be working with the Fibonacci set. This set is defined as:

$$\{1, 1, \text{and } x_n = x_{n-1} + x_{n-2}\} \text{ for } n = 3 \rightarrow \infty$$

We won’t work with **all** of them, so we’ll just use 1,1,2,3,5,8 for starters.

Create a list containing this shortened sequence and name it l (small L).

Create a set containing this shortened sequence and name it s.

Create a dictionary named d with the named items and their values, so

d = {“first”: 1, “second”:1, “third”:2, …}

Using these three data structures, implement some code that outputs the text below.

Note:

- The name of type is found type(l).__name__.
- You should NOT have to use the len() function. You can use negative indexes in Python. This is useful for computing the statement ‘the next number in the sequence is …’ without hard coding numbers or indexes.

collections.py output:

```
Working with collections:


First, let's create some collections.

l is a list, values = [1, 1, 2, 3, 5, 8]

d is a dict, values {'second': 1, 'sixth': 8, 'fourth': 3, 'first': 1,
'fifth': 5, 'third': 2}

s is a set, values {8, 1, 2, 3, 5}


The next number in the sequence is 8+5 = 13


Is 'seventh' in the dictionary?

False

let's add it...

Is 'seventh' in the dictionary?

True, the value is 13.


Let's add some items to the set, s: adding 13 =>

{1, 2, 3, 5, 8, 13}

try adding it a few more times

{1, 2, 3, 5, 8, 13}
```

Notice that the way you defined the set and dictionary are not necessarily how they are display.

Part 4 – Slicing collections

Begin this final section by adding a new python file to your project named slice_and_dice.py. We are going to use slicing to answer some questions about a paragraph. The text we are working with is:

Today it is the beginning of spring. Let's rejoice, the sun is out!

Using just string.find() and slices, implement a program which outputs the following (note you can assume the period terminates the first sentence and that spaces separate words):

slice_and_dice.py output:

```
Working with slices


Let's slice up some strings. Note that this would work on lists as well

All of the follow is generated using slices (e.g. text[0:index])

We will start with this string: Today it is the beginning of spring. Let's
rejoice, the sun is out!


The first word ends at index 5 and is Today.

The last sentence is: 'Let's rejoice, the sun is out!'

The last six characters of the string are: 's out!'

The first five characters of the second sentence are: 'Let's'
```