From algorithms to python

Principles of Computing (15-110)

Instructor: Giselle Reis

Lecture 04

Suppose you have invested a 100 Riyals on a fixed income fund that pays 0.2% per month. If you leave your money there for one month, how much do you have at the end? What about two months? What about three years?

Each time a month passes, you are paid 0.2% of the value in the fund. In the first month this value is 100, so at the end, you will have 100×1.002. In the second month, the amount in the fund is 100×1.002, so at the end of that month you will have $(100 \times 1.002) \times 1.002$. At the end of the third month, you will have $((100 \times 1.002) \times 1.002) \times 1.002$, and so on and so forth. You might have noticed the pattern now. At the end of the n-th month, you have 100×1.002^n.

In general, for an initial amount a in a fund paying i%, at the end of month n, the amount will be $a \times (1 + i/100)^n$. Can you write a step by step instruction on how to calculate the final amount, given the initial amount, the interest, and the number of months? (You may assume all mathematical operations, including exponentiation, are available.)

The steps written below make use of variables to compute each part of the formula separately.

1. Input: a (initial amount invested)
2. Input: i (interest – in percentage)
3. Input: n (number of months)
4. $j = i/100$
5. $b = 1 + j$
6. $f = b^n$
7. $r = a \times f$
8. The answer is r

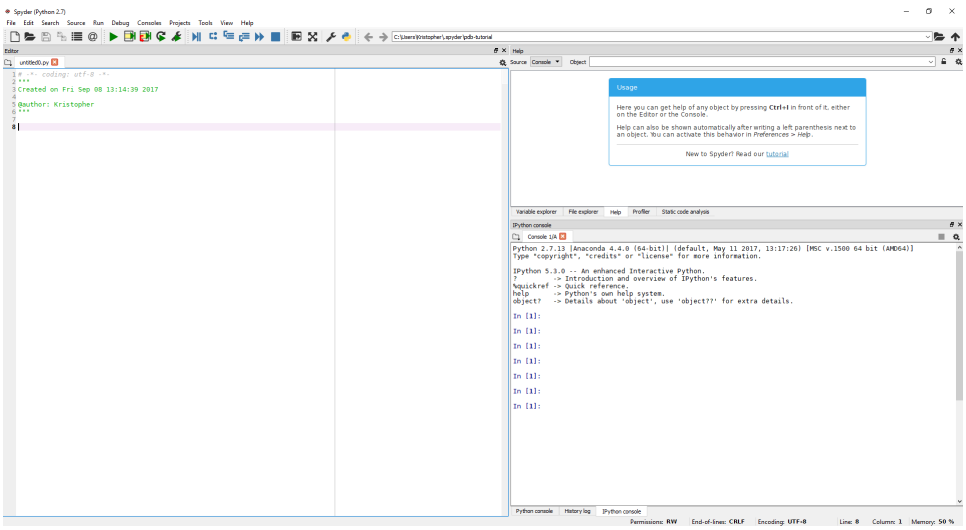
Once we come up with the solution, the sequence of instructions to solve the problem is quite simple. Let's see how we can write this in Python.

1 The IDE

IDE stands for *Integrated Development Environment*, and it is a program that is used for "program development", or programming. The IDE we will use for this course is called Spyder. If you have installed Anaconda correctly, you should have Spyder as well (they come together).

You can start Spyder either from the Anaconda Navigator (the thing that opens when you open Anaconda), or from the Start or Applications menu in your computer.

When the program opens, you might see something like this:



The window is divided into three parts.

- The left part is the *editor*, where you will open and edit python programs. This is where you type your code.
- The lower right part is the *console*. That is where you see error messages and where printed messages from your code will appear. You can also use it to test your code. Pay attention to this part when you run your program!
- The upper right part is actually three parts in one. You will see three tabs in its bottom edge which say: help, variable explorer, and file explorer. You don't have to worry about them for now, but feel free to explore if you'd like.

One of the most important buttons on the toolbar is the green "play" button. Clicking on it will cause the program in your editor to run, and its messages (if any) will be shown in the console. **You need to get into the habit of running your code as frequently as possible!** In this way, you fix the errors as you go, instead of accumulating and building on top of wrong code. It is helpful to know the keyboard shortcut for running your program: F5. That's because moving the hand out of the keyboard to the mouse/mousepad takes too long ;)

2 The Program

Now that you know more or less what is going on in the IDE, let's move to the actual programs. For the most part in this course, you will be given a *starter file* with a few things in it which you have to complete. For example, if the task at hand is the interest problem above, you might receive a python file containing:

```
1 #### interest ####
2
3 def interest(a, i, n):
4   return 42.0
```


You will open this file in Spyder.

The part after the # symbol will be a bit lighter than the rest and in italic font. That is because this is a comment. Comments are things in the code which are there for humans to better understand what is going on. The computer will ignore completely the comments. **It is useful to add comments to your code in order not to loose yourself.**

Typically, each task will ask you to implement one or more functions. A *function* is the block which starts with `def` and ends with `return`. Each function has a name, which is immediately after the word `def` and before the parenthesis. In the example above, the name of the function is `interest`. There's more to functions, but we will see this later in the course.

All the algorithms we have described so far in the course can be translated into a function. The inputs are the *arguments* or *parameters* of the function, which are the things inside the parentheses. If you compare the function above with the sequence of steps at the beginning of this lecture, you will see that both have three inputs. I used the same name for convenience.

The outputs of an algorithm is what the function is supposed to return. At the moment, the function above is returning always 42.0, which is obviously the wrong value. All the functions given to you will initially return a dummy value, which you have to change.

Your exercise will usually be to complete the function. That means you will add lines between `def interest(a, i, n):` and `return 42.0` to describe the sequence of steps you came up to solve the problem.

Let's do that for the interest problem above.

The input values are the ones inside the parentheses, so steps 1 to 3 are done.

Steps 4 to 7 involve the creation of new names to store intermediate values we calculate. New names can be simply introduced in python by typing the name, then the equal sign, and then the value it is supposed to be. These names that store values are called *variables*. Their names can be as long and as informative as you want.

Therefore, we can declare variables and *assign* values to them in python like this:

```
#### interest ####

def interest(a, i, n):
  j = i / 100
  b = 1 + j
  f = b ** n
  r = a * f
  return 42.0
```

The operation `/` is the division, `+` is addition, `**` is exponentiation, and `*` is multiplication.

The last thing we need to do is change the `return` statement to the thing that is the actual answer of the problem, which is r.

```
#### interest ####

def interest(a, i, n):
  j = i / 100
  b = 1 + j
  f = b ** n
  r = a * f
  return r
```

If you want, you can always add comments to your program so that you do not forget what you did on the next day. Remember that everything after a # is ignored by the computer. For example:


```
#### interest ####

def interest(a, i, n):
  j = i / 100
  b = 1 + j
  f = b ** n # This is exponentiation
  r = a * f
  return r
```

Attention about whitespaces!

Python is a very picky language when it comes to whitespaces. You will notice that if you do not align all the lines above exactly in the same column, Spyder will show a little warning sign on the left of the misaligned line. This space before each line is called *indentation* in programming.

Python uses the indentation to decide on the *scope* of an instruction. If a line is more to the right than the line above, it is inside its scope. If it is more to the left, it is outside its scope. For example, all of the lines below the `def` line in the code above are shifted to the right, because all of these instructions are *inside* the function.

You can always use the Tab or Shift+Tab keys to move your lines to the right or to the left.

Tips

- Use Tab for autocompleting commands and variables names in the code
- Use F5 for running the program
- Run the program often
- Keep an eye on the console, it informs you which line is the error

3 Testing

We wrote the program, now what? Now we need to run it and see if it works.

There are various ways to test a program. I suggest you start with the console. Once you hit F5, you will see that something changes on the console. It will try to run your file. If there is an error, it will show this error and you need to fix it before going on. If there is no error, then you can *call* your functions to see if they return what is expected.

To call a function, you will type its name, followed by some inputs in parentheses in the console. If we want to test the interest code for an initial amount of 100 Riyals, an interest of 2%, and a duration of one month, we would write:

```
In[X]: interest(100, 2, 1)
```

Note that this `X` may be any number. This is unimportant.

My suggestion is to always try a simple test first, one you are sure of the answer.

After pressing Enter, the console should tell you the answer:

```
In[X]: interest(100, 2, 1)
Out[X]: 102.0
```

If this is what you expected, good! But you should not stop there. Try to find *corner cases*, i.e., what if one of the inputs is 0? What if it is a real number instead of an integer? We will be

testing your code with many cases, so try to make sure it is robust, it works for as many cases as you can think of.

Streamlining tests

If you want to avoid typing the tests each time, you can make use of `assert` statements. Place these after your function, and outside it. An `assert` will compare the result of your function with the expected result. If it fails, you will see an error in the console. For the function above, you can write, for example:

```
#### interest ####

def interest(a, i, n):
  j = i / 100
  b = 1 + j
  f = b ** n # This is exponentiation
  r = a * f
  return r

assert(interest(100, 2, 1) == 102.0)
assert(interest(100, 0.2, 1) == 100.2)
```

See what happens when you make a mistake in the function, or when you use a value for the assert that is not the result of the function.

4 Exercise

Now that you know how to translate algorithms into code, translate the following sequence of instructions into Python. The initial function is:

```
def quiz01(): # A function with no inputs
  return 42.0
```

1. Start with the number 7
2. Multiply by the current month
3. Subtract 1
4. Multiply by 13
5. Add today's day
6. Add 3
7. Multiply by 11
8. Subtract the current month
9. Subtract the current day
10. Divide by 10
11. Add 11
12. Divide by 100

