Programming

More Loops

Vassilis Markos, Mediterranean College

Week 03

Contents

Last Time’s Exercises

In–class Exercise #001

A pyflix subscription (fictional python tutorials streaming service) has the following pricing scheme:

- The first 5 tutorials are free.
- The next 10 tutorials cost 5$ each.
- The next 20 tutorials cost 4$ each.
- Any further tutorials cost 2.5$ each.

Write a Python program that asks the user the number of tutorials they want to attend and computes the corresponding total cost.

In–class Exercise #002

Rewrite the following without any `elif`, i.e., only using `if-else`:

```python
# source/exercise_002.py

n = int(input("Please, enter an integer: "))
if n > 10:
    print("What?")
elif n > 15:
    print("Why?")
elif n > 20:
    print("How?")
else:
    print("Well...")
```

In–class Exercise #003

A quadratic equation, $ax^2 + bx + c = 0$, $a \neq 0$, is solved using the following formula:

$$x = \frac{-b \pm \sqrt{\Delta}}{2a}, \quad \Delta = b^2 - 4ac.$$

if $\Delta > 0$. If $\Delta = 0$ then the equation has a single solution, $x = -\frac{b}{2a}$, while if $\Delta < 0$ the equation has no (real) roots.
Write a Python program that asks the user to provide the three coefficients of a quadratic equation, a, b, c, and prints its solution(s), if any, or an appropriate message if no solutions exist.

In–class Exercise #004

Assuming that you are allowed to use only division by (and modulo of) 2 and 3, write a python program that:

- Asks the user for an integer number, n.
- Prints on screen an appropriate message for all the following cases:
  - Whether n is a multiple of 2.
  - Whether n is a multiple of 3.
  - Whether n is a multiple of 6.
  - Whether n is a multiple of 24.
- Explain your rationale by providing appropriate comments in your code.

In–class Exercise #005

A number n divides a number m if `m % n == 0`. Write a Python program that asks the user for a positive integer n and prints on screen all of its divisors.
Your program **should check that the user indeed provides a positive integer**, i.e., it should ask the user to provide another number in case they provided a non–positive integer (ignore cases where user input might not be integer).

In–class Exercise #006

- We say that n is a proper divisor of m if n is a divisor of m and $n \neq m$. A positive integer is said to be **perfect** if it is equal to the sum of its proper divisors.
- Write a Python program that asks the user for a positive integer and checks if its is perfect or not, printing a relevant message on screen.
- Your program should check that the user indeed provides a positive integer as in the previous exercise.

In–class Exercise #007

- Two numbers are said to be **amicable** if each one is equal the sum of the proper divisors of the other.
- Write a Python program that asks the user for two positive integers and checks if they are amicable or not, printing a relevant message on screen.
- Your program should check that the user indeed provides a positive integer as in the previous exercise.

Loops (Revisited)

A Simple Program

What does the following Python program do?

```python
# source/loops_001.py

n = int(input("Please, enter an integer: "))
i = 0
s = 0
while i < n:
    s = s + i
    i = i + 1
print(s)
```

The `for` Loop

Python also offers another way to loop through things:

```python
# source/loops_002.py

n = int(input("Please, enter an integer: "))
s = 0
for i in range(n):
    s = s + i
print(s)
```

The `for` Loop

The general syntax of a `for` loop is:

```
for <variable> in <iterable>:
    <stuff to be repeated>
```

- In Python, iterables are things we can iterate through, which, for now, should more or less be `range`s.
- But, what is a `range`?

What Is A `range`?

The easiest way to learn about things in Python is to print them. So, what does the following print?

```python
# source/loops_003.py

r = range(10)
print(r)
print(type(r))
for x in r:
    print(x)
```

Ranges

In Python, ranges are:

- Objects (as everything) that represent a range of values.
- We can think of them as lists of values, but they are not actually lists, as they **generate** the next element upon request.
- We can use ranges to iterate over them using `for` loops, as we have just seen.

`range` Syntax

There are three ways to declare a range:

- `range(n)` creates a range with integer values from `0` (included) to `n` (excluded).
- `range(m, n)` creates a range with integer values from `m` (included) to `n` (excluded). If $m \geq n$ then an empty range is produced.
- `range(m, n, s)` creates a range with integer values from `m` (included) to `n` (excluded) with a step of `s`, i.e., it skips every `s` values. Again, $m \geq n$ results to an empty range for $s > 0$ while it works well for $s < 0$.

What Will This Print?

```python
# source/loops_004.py

n = int(input("Please, enter an integer: "))
for i in range(n):
    print(i ** 2)
```

What Will This Print?

```python
# source/loops_004.py

n = int(input("Please, enter an integer: "))
for i in range(n):
    print(i ** 2)
```

- It prints the squares of numbers from 0 to $n - 1$.

What Will This Print?

```python
# source/loops_004.py

n = int(input("Please, enter an integer: "))
for i in range(n):
    print(i ** 2)
```

- It prints the squares of numbers from 0 to $n - 1$.
- How can we make this prints the squares of numbers from 1 to n using a `for` loop?

An Easy Fix

One way could be as follows:

```python
# source/loops_005.py

n = int(input("Please, enter an integer: "))
for i in range(1, n + 1):
    print(i ** 2)
```

An Easy Fix

One way could be as follows:

```python
# source/loops_005.py

n = int(input("Please, enter an integer: "))
for i in range(1, n + 1):
    print(i ** 2)
```

How can we print the results in **reverse order?**

In Reverse Order

Remember, ranges can take a negative step:

```python
# source/loops_006.py

n = int(input("Please, enter an integer: "))
for i in range(n, 0, -1):
    print(i ** 2)
```

In Reverse Order

Remember, ranges can take a negative step:

```python
# source/loops_006.py

n = int(input("Please, enter an integer: "))
for i in range(n, 0, -1):
    print(i ** 2)
```

Be careful! The starting point of a range is included while the ending is excluded!

A `print()` Interlude

So far, the outputs of the above programs should look like the one below:

```
Please, enter an integer: 5
25
16
9
4
1
```

While correct, this is a bit ugly, as numbers are printed on separate rows

A `print()` Interlude

Python’s `print()` allows us to modify its behaviour through some keyword arguments (we shall explain deeply what a keyword argument is in upcoming lectures):

- The `end` argument allows us to determine what will be the last character appended after everything has been printed on screen. It defaults to a newline character.
- The `sep` argument allows us to determine which character is printed between the arguments provided to `print()` for printing. It defaults to a single space.

A `print()` Interlude

So, for instance, we can modify our last program as follows, to print all numbers on the same line:

```python
# source/loops_007.py

n = int(input("Please, enter an integer: "))
for i in range(n, 0, -1):
    print(i ** 2, end = ", ")
```

A `print()` Interlude

So, for instance, we can modify our last program as follows, to print all numbers on the same line:

```python
# source/loops_007.py

n = int(input("Please, enter an integer: "))
for i in range(n, 0, -1):
    print(i ** 2, end = ", ")
```

There are some formatting issues, though. How can we address them?

A `print()` Interlude

To begin with, we can add an empty print, which, by default, prints just a newline, so as to change line once we have print all numbers, as follows:

```python
# source/loops_008.py

n = int(input("Please, enter an integer: "))
for i in range(n, 0, -1):
    print(i ** 2, end = ", ")
print()
```

A `print()` Interlude

To begin with, we can add an empty print, which, by default, prints just a newline, so as to change line once we have print all numbers, as follows:

```python
# source/loops_008.py

n = int(input("Please, enter an integer: "))
for i in range(n, 0, -1):
    print(i ** 2, end = ", ")
print()
```

What about the trailing comma after the last printed number?

A `print()` Interlude

We can capture this case using an `if` as follows:

```python
# source/loops_009.py

n = int(input("Please, enter an integer: "))
for i in range(n, 0, -1):
    if i == 1:
        print(i ** 2)
    else:
        print(i ** 2, end = ", ")
```

Python’s Inline `if-elif-else`

Python also offers a way to write simple `if` statements in a single line (much like the ternary operator in other languages):

```python
# source/loops_010.py

n = int(input("Please, enter an integer: "))
for i in range(n, 0, -1):
    print(i ** 2, end = ", " if i > 1 else "\n")
```

What Will This Print?

```python
# source/loops_011.py

n = int(input("Please, enter an integer: "))
s = 0
for i in range(1, n + 1):
    s = s + 1 / i ** 2
print(s)
```

What Will This Print?

```python
# source/loops_011.py

n = int(input("Please, enter an integer: "))
s = 0
for i in range(1, n + 1):
    s = s + 1 / i ** 2
print(s)
```

It prints the following sum:

$$s = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \cdots + \frac{1}{n^2}.$$

Odd Only?

How can we compute the sum of the odd terms, only, i.e.:

$$s = \frac{1}{1^2} + \frac{1}{3^2} + \frac{1}{5^2} + \frac{1}{7^2} + \cdots$$

Odd Only?

How can we compute the sum of the odd terms, only, i.e.:

$$s = \frac{1}{1^2} + \frac{1}{3^2} + \frac{1}{5^2} + \frac{1}{7^2} + \cdots$$

```python
# source/loops_012.py

n = int(input("Please, enter an integer: "))
s = 0
for i in range(1, n + 1, 2):
    s = s + 1 / i ** 2
print(s)
```

Even Only?

How can we compute the sum of the even terms, only, i.e.:

$$s = \frac{1}{2^2} + \frac{1}{4^2} + \frac{1}{6^2} + \frac{1}{8^2} + \cdots$$

Even Only?

How can we compute the sum of the even terms, only, i.e.:

$$s = \frac{1}{2^2} + \frac{1}{4^2} + \frac{1}{6^2} + \frac{1}{8^2} + \cdots$$

```python
# source/loops_013.py

n = int(input("Please, enter an integer: "))
s = 0
for i in range(2, n + 1, 2):
    s = s + 1 / i ** 2
print(s)
```

`while` Loops To `for` Loops

Write the following program using no `while` loops:

```python
# source/loops_015.py

n = int(input("Please, enter an integer: "))
i = 1
p = 1
while i < n:
    p = p * (1 / i ** 0.5)
    i = i + 3
print(p)
```

while Loops To for Loops

One solution could be as follows:

```python
# source/loops_016.py

n = int(input("Please, enter an integer: "))
p = 1
for i in range(1, n, 3):
    p = p * (1 / i ** 0.5)
print(p)
```

while Loops To for Loops

Write the following program using no while loops:

```
# source/loops_017.py

n = int(input("Please, enter an integer: "))
i = 2
while i < n and n % i != 0:
    i = i + 2
print(i)
```

`while` Loops To `for` Loops

One solution could be as follows:

```python
# source/loops_018.py

n = int(input("Please, enter an integer: "))
for i in range(2, n + 2, 2):
    if n % i == 0:
        break
print(i)
```

`break`

In Python, `break` is used to stop the execution of a loop and break out of it.

- All code in the same block and below a `break` won't be executed.
- Using `break` we can write any code we would write using `while`, i.e., capture more complex termination conditions.
- As above, observe that some times the range limits when transitioning from `while` to `for` loops have to be carefully adjusted!

What Will This Print?

```python
# source/loops_014.py

n = int(input("Please, enter an integer: "))
m = int(input("Please, enter an integer: "))
for i in range(n):
    for j in range(m):
        print(i, j, sep = "-", end = " ")
    print()
```

Printing Grids

For inputs 4 and 6 this should print something like the following:

```
0-0  0-1  0-2  0-3  0-4  0-5
1-0  1-1  1-2  1-3  1-4  1-5
2-0  2-1  2-2  2-3  2-4  2-5
3-0  3-1  3-2  3-3  3-4  3-5
```

Can you explain why?

Nested Loops

Nesting loops results to the inner loops being repeated so many times as the external loop dictates. For instance:

```
for i in range(10):
    for j in range(20):
        <stuff>
```

will repeat `<stuff>` $10 \times 20 = 200$ times, since:

- the inner loop repeats `<stuff>` 20 times, and;
- the outer loop repeats the inner loop 10 times.

So, it is important to remember that nesting loops leads to **multiplicatively longer** execution times!

Fun Time!

In–class Exercise #001

Write a Python program that computes the following sum:

$$s = 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \cdots + \frac{1}{2^n},$$

where n is provided by the user.
Then, make any modifications needed to compute the following sum:

$$s = 1 - \frac{1}{2} + \frac{1}{4} - \frac{1}{8} + \frac{1}{16} - \cdots \pm \frac{1}{2^n}.$$

In–class Exercise #002

Write a Python program that asks the user for a positive integer, n, and computes its factorial, denoted by $n!$. As a reminder, the factorial of a number, n, is given by the following formula:

$$n! = n \cdot (n-1) \cdot (n-2) \cdots 2 \cdot 1.$$

So, for instance:

$$\begin{aligned} 3! &= 3 \cdot 2 \cdot 1 = 6, \\ 5! &= 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120, \\ 8! &= 8 \cdot 7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 40320. \end{aligned}$$

In–class Exercise #003: Part A

In combinatorics, a quite useful quantity is the number of subsets of k elements from a set of n elements, with $k < n$, given by the following formula:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}.$$

Using code from your previous exercise, write a Python program that asks the user for n, k and computes $\binom{n}{k}$ (which is often called the **binomial coefficient**).

In–class Exercise #003: Part B

Test your previous program on the following input:

$$n = 10000, \quad k = 5000.$$

- How did it perform?
- Can you find a way to fix it? You can, of course, look around the web for ideas.
- Explain the rationale of your solution.

In–class Exercise #004: Part A

A positive integer number, n, is said to be prime if:

- $n > 1$, and;
- the only divisors of n are 1 and n.

Explain why the following program, intended to check whether a number n is prime is wrong:

```python
# source/exercise_004a.py

n = int(input("Please, enter an integer: "))
if n > 1 and n % n ==0 and n % 1 == 0:
    print("Prime!")
else:
    print("Not Prime!")
```

In–class Exercise #004: Part B

A positive integer number, n, is said to be prime if:

- $n > 1$, and;
- the only divisors of n are 1 and n.

Write a Python program that asks the user for a positive integer and computes whether it is prime or not. Test your program on the following inputs:

$$1, 2, 3, 4, 5, 7, 9, 14, 23, 57, 101.$$

The outputs should be (`True` for prime, `False` for non–prime):

`False`, `True`, `True`, `False`, `True`, `True`, `False`, `False`, `True`, `False`, `True`.

In–class Exercise #004: Part C

Test your previous program about primes on the following input:

$$1234567891.$$

- How long did it take to terminate?
- Can you make it run faster?
- How long does your new implementation take on 10101012107?

In–class Exercise #005

Write a Python function that asks the user for the number of rows and columns and prints a shape like what is shown next on screen (this is for 5 rows and 6 columns).

```
*  *  *  *  *  *

*  *  *  *  *  *

*  *  *  *  *  *

*  *  *  *  *  *

*  *  *  *  *  *
```

In–class Exercise #006

Write a Python function that asks the user for the number of rows / columns (it should be the same in this case) and prints a shape like what is shown next on screen (this is for 5 rows / columns).

```
*  *  *  *  *

*  *  *  *

*  *  *

*  *

*
```

In–class Exercise #007

Write a Python function that asks the user for the number of rows / columns (it should be the same in this case) and prints a shape like what is shown next on screen (this is for 5 rows / columns).

```
            *
         *  *
      *  *  *
   *  *  *  *
*  *  *  *  *
```

In–class Exercise #008

Write a Python function that asks the user for the number of rows / columns (it should be the same in this case) and prints a shape like what is shown next on screen (this is for 5 rows / columns).

```
*
*
*
*
* * * * *
```

In–class Exercise #009

Write a Python function that asks the user for the number of rows / columns (it should be the same in this case) and prints a shape like what is shown next on screen (this is for 5 rows / columns).

```
*   *   *   *   *

*               *

*               *

*               *

*   *   *   *   *
```

In–class Exercise #010

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