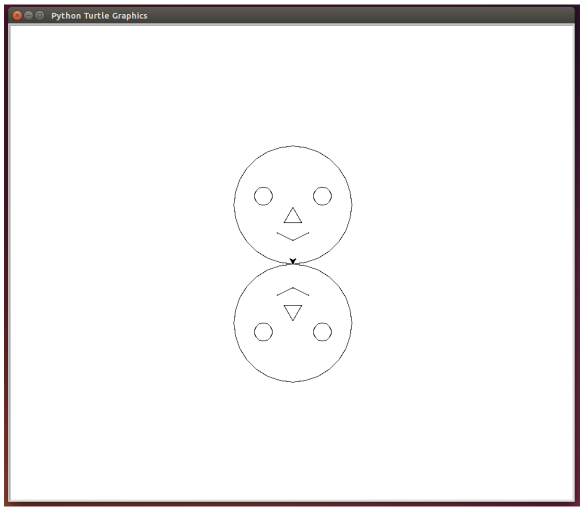
# Informatike I — CSCI-141
# Vizatimet e breshkes — Leksion 1

08/17/2017

## 1 Pershkrimi i problemit



Nje menyre per te bere vizatime te thjeshta eshte duke perdorur librarine `turtle` ne Python. Si fillim, le te shikojme si mund te vizatojme nje fytyre. Per te nenvizuar ”riperdorimin” e programeve, ne kete rast do vizatojme dy fytyra.

## 2 Analiza dhe projekti i zgjidhjes

Komandat e perdorura jane pjese te librarise turtle. Breshka (turtle) eshte nje objekt imagjinar qe leviz ne hapesiren e ekranit dhe ka ne dore nje laps. Gjate levizjes breshka mund te vizatoje figura te ndryshme. Disa nga veprimet jane:

- ecje para per nje distance te caktuar;
- levizje rrethore (ne drejtim te kundert me akrepat e ores);
- rrotullim majtas (djathtas) me nje numer te caktuar gradesh;
- ulje/ngritje e lapsit. Kur lapsi eshte poshte, cdo levizje e breshkes vizatohet ne ekran.

Projekti i zgjidhjes perfshin keto aktivitete:

- Percaktimi i nje pozicioni fillestar (home) per breshken.
- Pershkrim ne menyre algoritmike i levizjeve qe i duhen breshkes per te vizatuar figuren e kerkuar dhe per tu kthyer ne pozicionin fillestar.

**Pozicioni fillester**: Mesi i ekranit, me breshken qe shikon drejt veriut (V) dhe me lapsin e ngritur lart.

Duke percaktuar nje pozicion fillestar eshte me e lehte te shtosh ose te heqesh pjese nga figura.

## 2.1 Algorithms

An *algorithm* is a special set of instructions. In the words of David Berlinski[1],

```
An algorithm is

a finite procedure,

written in a fixed symbolic vocabulary,

governed by precise instructions,

moving in discrete steps, 1, 2, 3, ...,

whose execution requires no insight, cleverness,
intuition, intelligence, or perspicuity,

and that sooner or later comes to an end.
```

Below we define two algorithms. One will put the turtle in its home position, and the other will draw the entire face. Each algorithm is decomposed into a series of smaller steps. Executing the whole program consists of calling a procedure to execute the first algorithm, followed by calling a procedure to execute the second algorithm.

**Initialization Algorithm**

- move turtle to home position
- rotate turtle north
- lift pen up

**Face-Drawing Algorithm**

- draw the outline of the face
- draw the mouth
- draw the nose
- draw the eyes

## 2.2 Implementation

We will see that for this program each step of the Initialization algorithm can be implemented using a built-in command from the turtle drawing package. The steps of the Face-Drawing algorithm have a more elaborate implementation and need to be expanded when written in Python code. Each 'draw' command of the Face-Drawing algorithm exists as a separate function, or procedure, in the solution file.

1. **Advent of the Algorithm**. Harcourt. ISBN 0156013916 / 9780156013918 / 0-15-601391-6.

It's important to have documentation on the language and the modules we need to use to solve this problem. There are a number of links to Python language documentation on the course's resources page, `http://www.cs.rit.edu/~csci141/resources.html` You may find the "Summary Sheet" a good place to start. Information on the functions and procedures of the turtle module is at `http://docs.python.org/py3k/library/turtle`

See the accompanying source file, `smiling_faces2_py.txt`. *Note: The file has a* `.txt` *suffix because browsers may try to execute Python source files.*

Note: When the turtle software is loaded, the turtle is already at the location we desire for this program, and it is facing East. All we need to do is orient the turtle correctly and pull its pen up.

## 3 Testing (Test Cases, Procedures, etc.)

Large software programs have a practically infinite number of different ways they might execute, depending on input supplied to them. It is quite an art to choose a small set of tests that cover all the functionality in the software. Since this program is very small, this is not hard to do. Here are the two test cases.

1. Does a single face get drawn correctly?
   Call the Face-Drawing procedure with the turtle at its starting position.
2. Does it draw a face correctly, no matter where the turtle starts, and no matter how many times the face-drawing procedure is executed?
   Change the turtle's position and orientation and call the Face-Drawing procedure again; the program should draw a new face at that new location and orientation.

## 4 Learning Objectives

There is a lot in this lecture, especially for students with no experience (and instructors who are new to the course).

Students should learn:

1. the details of the course and its operation (web page, syllabus, mycourses);
2. Python function definition and interpretation of '`.py`' files (and at the '>>>' prompt);
3. the concepts of computational problem-solving using an example problem; and
4. How to approach and solve a specific problem (drawing faces) using Python.

### 4.1 Learned by the End of the Lecture

By the end of the lecture, the student should be able to:

- Start the Python interpreter, and quit the Python interpreter to terminate.
- Import the Python turtle module using either `import turtle` or `from turtle import *`. (While the first requires more typing, it is preferred because it enforces the notion of calling a function on an object, which is useful later.)
- Create a `.py` file containing functions with documentation.
- Run a `.py` file to execute its main function.
- Look up information on built-in Python functions and modules.
- Describe simple program flow (statement execution).
- Define and then invoke a parameter-less function.
- Describe the requirement of proper indenting and the problem of tabs versus spaces.
- State that they always need to use Python version 3 (the names end in "3" in CS department *NIX machines).
- Use the `input` and output (i.e. `print`) commands.

### 4.2 Logistics to Learn by the End of the Lecture

At the end of lecture, make sure the students know the following:

- Where the homework assignment is located, and when it must be uploaded;
- When and where to go for the lab session;
- When and where to go for the recitation session; and
- How they will get their CS department accounts (i.e. from the SLI in lab).

### 4.3 Problem-specific Learning Details

- General problem solving techniques
- Graphics programming using a line drawing package
- Pen control routines
  - Setting the pen position
  - Raising and lowering the pen

  - Moving and turning the pen
  - Waiting with the `done` versus the `input` commands.
- Testing

### 4.4 Learning Activities

1. Describe the course structure, organization and materials.
2. Introduce the Python programming language and development environment.
3. Introduce a Computational Problem-solving Approach.
4. Practice the approach on a specific problem (drawing faces).

The instructor should present the problem in the following order:

- Draw an approximation of the desired figure on the board.
- Go over the Python code (at the procedure level) or develop it live in class.
- List the operations needed, and show their online documentation. Here are some ways:
  - Use a browser to search and include the search text '*python 3*' so that you will get results for Python version 3 instead of version 2.
  - Use `pydoc3 turtle` in a shell.
  - `help( turtle )` in the interpreter (after `import turtle`.
  - Or visit `http://docs.python.org/py3k/library/turtle`.
- Ask students to develop the subroutines for drawing different parts of the figure.
- Introduce the Python language (version 3) and the turtle package.
- Go over the implementation or develop it live in class.
- Introduce either the editor **PyCharm**, **idle** (version 3) or **vim**, based on (a) whether or not the class uses a PC-based lab and (b) instructor's personal preference.
- Verify the solution using the first test case.
- Run the second test case, which draws a second figure at a new position and orientation; all the functions still work properly.