# Programming Fundamentals

## Debugging

João Correia Lopes

INESC TEC, FEUP

20 december 2018

# GOALS

By the end of this class, the student should be able to:

- Identify the kind of errors that may occur when writing programs
- Describe techniques to debug programs with errors
- Inserting breaking points and inspecting variables

# BIBLIOGRAPHY


- Peter Wentworth, Jeffrey Elkner, Allen B. Downey, and Chris Meyers, *How to Think Like a Computer Scientist — Learning with Python 3 (RLE)*, 2012 (Appendix A) [HTML]
- Spyder docs, *Debugging*, 2018 [HTML]

# TIPS

- There's no slides: we use a script, illustrations and code in the class. Note that this PDF is NOT a replacement for **studying the bibliography** listed in the *class plan*
- "Students are responsible for anything that transpires during a class—therefore **if you're not in a class**, you should get notes from someone else (not the instructor)"—David Mayer
- The best thing to do is to **read carefully** and **understand** the documentation published in the Content wiki (or else **ask** in the recitation class)
- We will be using **Moodle** as the primary means of communication

# CODE, TEST & PLAY

- Have a look at the code in GitHub:
  **https://github.com/fpro-admin/lectures/**
- Test before you submit at FPROtest:
  **http://fpro.fe.up.pt/test/**
- Pay a visit to the playground at FPROplay:
  **http://fpro.fe.up.pt/play/**

# CONTENTS

# KINDS OF ERRORS

Different *kinds of errors* can occur in a program, and it is useful to distinguish among them in order to track them down more quickly:

1. **Syntax errors** are produced by Python when it is translating the source code into byte code
2. **Runtime errors** are produced by the runtime system if something goes wrong while the program is running
3. **Semantic errors** are problems with a program that compiles and runs but doesn't do the right thing

# KINDS OF ERRORS (DETAILS)

1. **Syntax errors** are produced by Python when it is translating the source code into byte code
   - They usually indicate that there is something wrong with the syntax of the program
   - Example: Omitting the colon at the end of a def statement yields the somewhat redundant message `SyntaxError: invalid syntax`
2. **Runtime errors** are produced by the runtime system if something goes wrong while the program is running
   - Most runtime error messages include information about where the error occurred and what functions were executing
   - Example: An infinite recursion eventually causes a runtime error of maximum recursion depth exceeded
3. **Semantic errors** are problems with a program that compiles and runs but doesn't do the right thing
   - Example: An expression may not be evaluated in the order you expect, yielding an unexpected result

# KINDS OF ERRORS (DETAILS)

1. **Syntax errors** are produced by Python when it is translating the source code into byte code
   - They usually indicate that there is something wrong with the syntax of the program
   - Example: Omitting the colon at the end of a def statement yields the somewhat redundant message `SyntaxError: invalid syntax`
2. **Runtime errors** are produced by the runtime system if something goes wrong while the program is running
   - Most runtime error messages include information about where the error occurred and what functions were executing
   - Example: An infinite recursion eventually causes a runtime error of maximum recursion depth exceeded
3. **Semantic errors** are problems with a program that compiles and runs but doesn't do the right thing
   - Example: An expression may not be evaluated in the order you expect, yielding an unexpected result

# KINDS OF ERRORS (DETAILS)

1. **Syntax errors** are produced by Python when it is translating the source code into byte code
   - They usually indicate that there is something wrong with the syntax of the program
   - Example: Omitting the colon at the end of a def statement yields the somewhat redundant message `SyntaxError: invalid syntax`
2. **Runtime errors** are produced by the runtime system if something goes wrong while the program is running
   - Most runtime error messages include information about where the error occurred and what functions were executing
   - Example: An infinite recursion eventually causes a runtime error of maximum recursion depth exceeded
3. **Semantic errors** are problems with a program that compiles and runs but doesn't do the right thing
   - Example: An expression may not be evaluated in the order you expect, yielding an unexpected result

# FIRST STEP IN DEBUGGING

- The first step in debugging is to figure out which kind of error you are dealing with
- Although the following sections are organized by error type, some techniques are applicable in more than one situation

# SYNTAX ERRORS

- Syntax errors are usually easy to fix once you figure out what they are
- Here are some ways to avoid the most common syntax errors (RLE)
- I can't get my program to run no matter what I do (RLE)

# RUNTIME ERRORS

- Once your program is syntactically correct, Python can import it and at least start running it
- What could possibly go wrong?
  - My program does absolutely nothing (RLE)
  - My program hangs (RLE)
  - Infinite Loop (RLE)
  - Infinite Recursion (RLE)
  - Flow of Execution (RLE)
  - When I run the program I get an exception (RLE)
  - I added so many `print` statements I get inundated with output (RLE)

# SEMANTIC ERRORS

- In some ways, semantic errors are the hardest to debug, because the compiler and the runtime system provide no information about what is wrong
- Only you know what the program is supposed to do, and only you know that it isn't doing it
- The first step is to make a connection between the program text and the behavior you are seeing
  - My program doesn't work (RLE)
  - I've got a big hairy expression and it doesn't do what I expect (RLE)
  - I've got a function or method that doesn't return what I expect (RLE)
  - I'm really, really stuck and I need help (RLE)
  - No, I really need help (RLE)
- The time it takes to insert a few well-placed `print` statements is often short compared to setting up the debugger, inserting and removing breakpoints, and walking the program to where the error is occurring

# SEMANTIC ERRORS

- In some ways, semantic errors are the hardest to debug, because the compiler and the runtime system provide no information about what is wrong
- Only you know what the program is supposed to do, and only you know that it isn't doing it
- The first step is to make a connection between the program text and the behavior you are seeing
  - My program doesn't work (RLE)
  - I've got a big hairy expression and it doesn't do what I expect (RLE)
  - I've got a function or method that doesn't return what I expect (RLE)
  - I'm really, really stuck and I need help (RLE)
  - No, I really need help (RLE)
- The time it takes to insert a few well-placed `print` statements is often short compared to setting up the debugger, inserting and removing breakpoints, and walking the program to where the error is occurring

# How to be a Successful Programmer (Recap)

- One of the most important skills you need to aquire is the ability to debug your programs
- Debugging is a skill that you need to master over time
- As programmers we spend 99% of our time trying to get our program to work
- But here is the secret, when you are successful, you are happy, your brain releases a bit of chemical that makes you feel good
- **Start small, get something small working, and then add to it**

# How to Avoid Debugging (Recap)

## Mantra

Get something working and keep it working

- **Start Small**
  - This is probably the single biggest piece of advice for programmers at every level.
- **Keep it working**
  - Once you have a small part of your program working the next step is to figure out something small to add to it.

Ok, let's look at an example: the `alarm_clock.py` of RE02

⇒ https://github.com/fpro-admin/lectures/blob/master/06/alarm_clock.py

# Beginning tips for Debugging (Recap)

Debugging a program is a different way of thinking than writing a program.
The process of debugging is much more like being a detective.

1. Everyone is a suspect (Except Python)!
2. Find clues
   - Error messages
   - Print statements

# Beginning tips for Debugging (Recap)

Debugging a program is a different way of thinking than writing a program.
The process of debugging is much more like being a detective.

1. Everyone is a suspect (Except Python)!
2. Find clues
   - Error messages
   - Print statements

# Summary on debugging (Recap)

- Make sure you take the time to understand error messages
  - They can help you a lot
- `print` statements are your friends
  - Use them to help you uncover what is **really** happening in your code
- Work backward from the error
  - Many times an error message is caused by something that has happened before it in the program
  - Always remember that Python evaluates a program top to bottom

# DEBUGGING IN SPYDER

- Debugging in Spyder is supported through integration with the enhanced *ipdb debugger* in the IPython Console (Spyder | Docs)
- This allows **breakpoints** and the execution flow to be viewed and controlled right from the Spyder GUI, as well as with all the familiar IPython console commands
- Inspecting variables (Spyder | Docs)

# EXERCISES

- Moodle activity at: LE24: Debugging