University of HUDDERSFIELD

CFS2160: Software Design and Development

01: Some Basics

Methodology. Starting to explore Python.

Tony Jenkins
A.Jenkins@hud.ac.uk

Status Report

By now, you should have:

- Used PythonAnywhere, and run a program.
- Created a GitHub student account.
- Used PyCharm and run "Hello World".
- Got "The Book" and read the first chapter.

You might have:

- Set up PyCharm on your own computer.
- Started to explore more of Python.

Status Report

By now, you should have:

- Used PythonAnywhere, and run a program.
- Created a GitHub student account.
- Used PyCharm and run "Hello World".
- Got "The Book" and read the first chap

You might have:

- Set up PyCharm on your own compute
- Started to explore more of Python.

If you have not, you are behind and need to catch up.

Programming is not something you can catch up at the end!

Programming is a Skill

Programming is a *skill*.

Other *skills* are:

- Playing the piano.
- Driving.
- Cookery.
- Ballroom Dancing.
- Juggling.

Practice.

Practice More.

Then Practice Some More.

Programming is a Discipline

Good programming requires a *disciplined* approach.

It is possible to just "bash a solution together", but the results might:

- Not be as efficient as they could be.
- Have hidden, subtle, "bugs".
- Not deal with all possible cases.
- Be a nightmare to maintain and update.
- Be impenetrable to other programmers.
- Be just plain ugly.

Programming is a Discipline

Good programming requires a *disciplined* approach.

It is possible to just "bash a solution together", but the results might:

- Not be as efficient as they could be.
- Have hidden, subtle, "bugs".
- Not deal with all possible cases.
- Be a nightmare to maintain and updat
- Be impenetrable to other programmer
- Be just plain ugly.

Every problem has many possible solutions.

Some of these solutions are better than others.

"Methodology"

A structured approach to programming is called a "Methodology".

It usually runs something like:

1. Understand the Task.
   - ➢ What? Who? When? How Much?
2. Design a Solution.
   - ➢ Consider alternatives. Look at existing similar solutions. Choose the "Best".
3. Implement the Solution.
4. Test and Review the Solution.
   - ➢ Evaluate. Reflect. If necessary, return to an earlier stage.

Pizza

Here's an analogy:

Suppose we want to make a pizza. The process runs something like:

1. Understand.
   - ➢ What toppings? What base? When needed?
2. Design.
   - ➢ When to start? What to buy? How to cook?
3. Implement.
   - ➢ Prepare. Cook. Deliver.
4. Test.
   - ➢ Was it yummy? Could the next one be better?

Recipes

If you have ever cooked using a recipe from a cookbook, you probably know that they follow a format:

- Ingredients at the top.
- Then instructions, in order, usually numbered.

And these instructions:

- Use a certain type of language.
- Involve choices and decisions.
- Involve repetition.

Recipes

If you have ever cooked using a recipe from a cookbook, you probably know that they follow a format:

- ➢ Ingredients at the top.
- ➢ Then instructions, in order, usually numbered.

And these instructions:

- ➢ Use a certain type of language.
- ➢ Involve choices and decisions.
- ➢ Involve repetition.

It would (?) be possible to follow a recipe with no idea of what you were cooking.

But it would be much better to have a full understanding.

Recipes

If you have ever cooked using a recipe from a cookbook, you probably know that they follow a format:

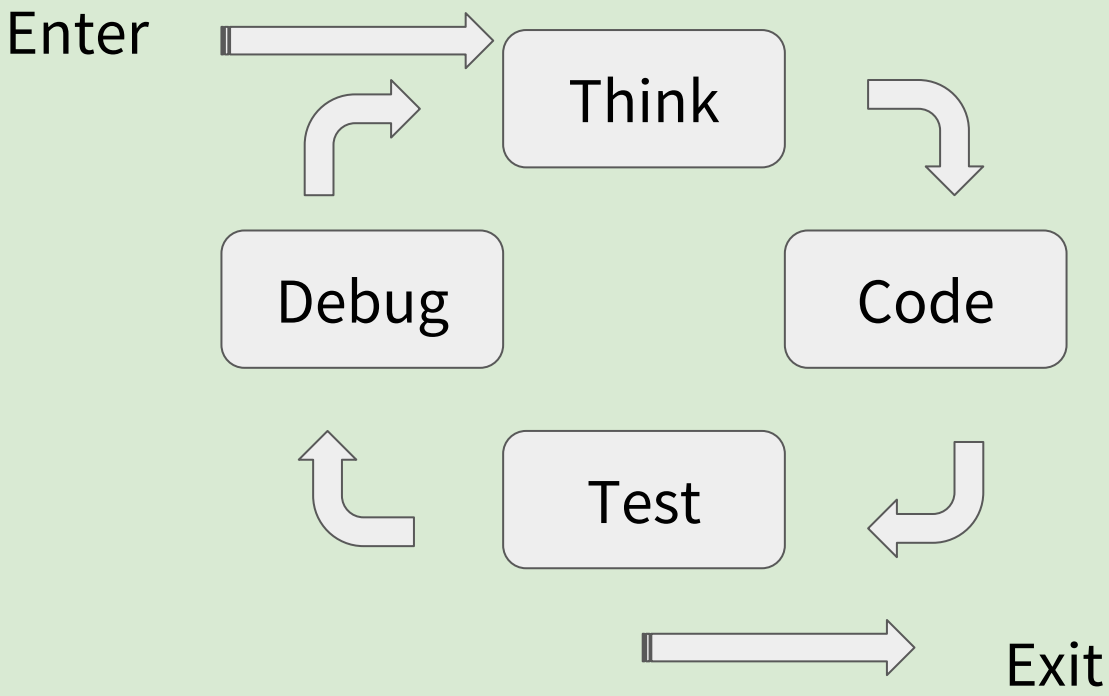
- ➢ Ingredients at the top.
- ➢ Then instructions, in order, usually numbered.

And these instructions:

- ➢ Use a certain type of language.
- ➢ Involve choices and decisions.
- ➢ Involve repetition.

Also, when you have followed a recipe to make one kind of pizza, you will be able to make new, different (better) pizzas, without looking up a recipe.

Think : Code : Test : Debug

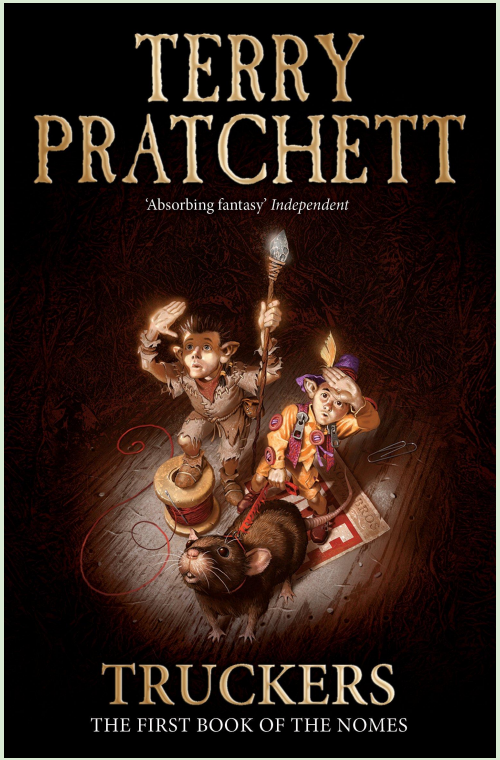


Breaking Down the Job

He knew what he had to do. It was, of course, an impossible task. But he was used to them.

Dragging a rat all the way from the wood to the hole had been an impossible task. But it wasn't impossible to drag it a little way, so you did that, and then you had a rest, and then you dragged it a little way again…

The way to deal with an impossible task was to chop it down into a number of merely very difficult tasks and break each one of them into a group of horribly hard tasks, and each one of them into tricky jobs, and each one of them ...



Specification

"We need some code that will check whether a customer can rent a car from us."

Specification

"We need some code that will check whether a customer can rent a car from us."

"OK. What are the rules that determine whether a customer can rent a car?"

Specification

"We need some code that will check whether a customer can rent a car from us."

"OK. What are the rules that determine whether a customer can rent a car?"

"They need to have a driving licence and be aged over 21."

Specification

"We need some code that will check whether a customer can rent a car from us."

"OK. What are the rules that determine whether a customer can rent a car?"

"They need to have a driving licence and be aged over 21."

"OK. Do you mean 21 or over? Are 21 year olds allowed?"

Specification

"We need some code that will check whether a customer can rent a car from us."

"OK. What are the rules that determine whether a customer can rent a car?"

"They need to have a driving licence and be aged over 21."

"OK. Do you mean 21 or over? Are 21 year olds allowed?"

"Ah, yes. 21 year olds are OK. And fewer than six points on the driving licence."

Specification

"We need some code that will check whether a customer can rent a car from us."

"OK. What are the rules that determine whether a customer can rent a car?"

"They need to have a driving licence and be aged over 21."

"OK. Do you mean 21 or over? Are 21 year olds allowed?"

"Ah, yes. 21 year olds are OK. And fewer than six points on the driving licence."

"OK. Understood."

Reusable Code

Most programs (if you think about it) do basically the same thing:

- Some stuff gets read from somewhere.
- Something is done to the stuff.
- The amended stuff gets written somewhere.

Reusable Code

Most programs (if you think about it) do basically the same thing:

- Some stuff gets read from somewhere.
- Something is done to the stuff.
- The amended stuff gets written somewhere.

It follows that many programs are made up of code that it basically the same code.

Such code is said to be *reusable.*

Reusable Code

Most programs (if you think about it) do basically the same thing:

- Some stuff gets read from somewhere.
- Something is done to the stuff.
- The amended stuff gets written somewhere.

It follows that many programs are made up of co

Such code is said to be *reusable.*

This is also why programming gets easier with *experience*.

You build up your own library of reusable code, techniques, and tricks.

Reusable Code

Most programs (if you think about it) do basically the same thing:

- Some stuff gets read from somewhere.
- Something is done to the stuff.
- The amended stuff gets written somewhere.

It follows that many programs are made up of co

Such code is said to be *reusable.*

This is also why programming gets easier with *experience*.

Stick at it, and your current self will be *amazed* at what you're writing in three years' time.

Reusable Code

Most programs (if you think about it) do basically the same thing:

- Some stuff gets read from somewhere.
- Something is done to the stuff.
- The amended stuff gets written somewhere.

It follows that many programs are made up of co

Such code is said to be *reusable.*

You might also see the term "Boilerplate Code".

This is code, often repeated, that appears in many places with little alteration.

Reusable Code

Most programs (if you think about it) do basically the same thing:

- Some stuff gets read from somewhere.
- Something is done to the stuff.
- The amended stuff gets written somewhere.

It follows that many programs are made up of co

Such code is said to be *reusable.*

You might also see the term "Boilerplate Code".

IDEs like PyCharm and IntelliJ are really, really good at removing the pain of Boilerplate Code.

Reusable Code

"Reusability" in code also has implications for how code it written.

There is some code over there.

What is it for? What does it do?

```
def gets (x, y):

    z = x > 65;

    return z and y
```

Reusable Code

"Reusability" in code also has implications for how code it written.

There is some code over there.

What is it for? What does it do?

Naming things sensibly helps a lot.

```
def free_drugs (age, has_condition):

    if age > 65 and has_condition:
        return True
    else:
        return False
```

Reusable Code

"Reusability" in code also has implications for how code it written.

There is some code over there.

What is it for? What does it do?

Naming things sensibly helps a lot.

And ***comments*** can be added to clear up any possible confusion.

```
''' Drugs are free if patient is
    aged over 65 and has a
    long-standing condition.      '''

def free_drugs (age, has_condition):

    if age > 65 and has_condition:
        return True
    else:
        return False
```

Reusable Code

"Reusability" in code also has implications for how code it written.

There is some code over there.

What is it for? What does it do?

Naming things sensibly helps a lot.

And comments can be added to clear up any possible confusion.

Code can also be ***refactored*** for brevity and clarity.

```
''' Drugs are free if patient is
    aged over 65 and has a
    long-standing condition.      '''

def free_drugs (age, has_condition):

    return age > 65 and has_condition
```

Why Python?

Remember, we are learning to *program*.

Python is the first language we will use. Why?

- It is a "real world" language, used by many big organisations.
- It has an English-like syntax.
- It is possible to do cool stuff with very little code.
- There are many, many, free resources to help learners.
- It has an *interactive console*.

Python Console

The console can be activated:

- At the bottom of the PyCharm IDE.
  - (Precisely where depends a lot on settings. Look for the Python logo.)
- From the Tools menu of the PyCharm IDE.
- From the command line.
  - `$ python3`
- From the Windows "Start" Menu.
- Online, for example:
  - https://www.python.org/shell/
  - https://www.pythonanywhere.com/

Python Console

The console can be activated:

- At the bottom of the PyCharm IDE.
  - (Precisely where depends a lot on settings. Look for the Python logo.)
- From the Tools menu of the PyCharm IDE
- From the command line.
  - `$ python3`
- From the Windows "Start" Menu.
- Online, for example:
  - https://www.python.org/shell/
  - https://www.pythonanywhere.com/

Note

You may see a slightly different console, called IPython.
https://ipython.org/

Python Console


```
$ python3
Python 3.5.2 (default, Nov 17 2016, 17:05:23)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

Python Console

```
$ python3
Python 3.5.2 (default, Nov 17 2016, 17:05:23)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

The console is now waiting for Python commands.
We can use it to experiment, to test code, or just as a handy calculator.

Python Console

```
$ python3
Python 3.5.2 (default, Nov 17 2016, 17:05:23)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

The version of Python that is running is shown here.
We want something in the 3.x family.
Current version is 3.7.0, but anything above 3.5 is fine.

Python Console


```
$ python3
Python 3.5.2 (default, Nov 17 2016, 17:05:23)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

These three chevrons are often used in tutorials and docs to denote the Python console.

Python Console


```
$ python3
Python 3.5.2 (default, Nov 17 2016, 17:05:23)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>> eggs = 14
>>> boxes = eggs // 6
>>> left_over = eggs - (boxes * 6)
>>> print (boxes)
2
>>> print (left_over)
2
```

Python Variables and Values

A *variable* is *declared* by assigning it a value.

```
>>> eggs = 14
```

Python Variables and Values

A *variable* is *declared* by assigning it a value.

```
>>> eggs = 14
```

A variable has a *type*, which is determined by the value assigned.

```
>>> type (eggs)
<class 'int'>
```

Python Variables and Values

A *variable* is *declared* by assigning it a value.

A variable has a *type*, which is determined by the value assigned.

If used as an *argument* to the `print` command, the value will be output.

```
>>> eggs = 14

>>> type (eggs)
<class 'int'>

>>> print (eggs)
14
```

Python Variables and Values

A *variable* is *declared* by assigning it a value.

A variable has a *type*, which is determined by the value assigned.

If used as an *argument* to the `print` command, the value will be output.

A variable can be used in an *expression*. Here we also create a new variable (% is "modulus", or "remainder").

```
>>> eggs = 14

>>> type (eggs)
<class 'int'>

>>> print (eggs)
14


>>> left_over = eggs % 6
>>> print (left_over)
```

Python Variables and Values

Simple data types are:

- Integers (see previous code).
- Strings.
- Floating-point numbers.
- Booleans (`True` or `False`).

```
>>> choice = 'Y'
>>> type (choice)
<class 'str'>

>>> interest_rate = 0.15
>>> type (interest_rate)
<class 'float'>

>>> has_licence = True
>>> type (has_licence)
<class 'bool'>
```

Python Variables and Values

Simple data types are:

- Integers (see previous code).
- Strings.
- Floating-point numbers.
- Booleans (`True` or `False`).

Remember that these are all different types:

- 1
- "1"
- 1.0

```
>>> choice = 'Y'
>>> type (choice)
<class 'str'>

>>> interest_rate = 0.15
>>> type (interest_rate)
<class 'float'>

>>> has_licence = True
>>> type (has_licence)
<class 'bool'>
```

Python Operators

Operators combine values to give new values.

Python Operators

Operators combine values to give new values.

```
>>> 2 + 2
4
```

Python Operators

Operators combine values to give new values.

These values can be taken from variables.

```
>>> 2 + 2
4
>>> eggs = 11
>>> eggs * 2
22
```

Python Operators

Operators combine values to give new values.

```
>>> 2 + 2
4
```

These values can be taken from variables.

The results can be stored in other variables.

```
>>> eggs = 11
>>> boxes = eggs // 6
>>> boxes
1
```

Python Operators

Operators combine values to give new values.

These values can be taken from variables.

The results can be stored in other variables.

The common operators are:

- \+
- \-
- *
- / and // and %

```
>>> 2 + 2
4
```

```
>>> eggs = 11
>>> boxes = eggs // 6
>>> boxes
1
```

Python Operators

Operators combine values to give new values.

These values can be taken from variables.

The results can be stored in other variables.

The common operators are:

- \+
- \-
- *
- **/ and // and %**

```
>>> 5 / 2
2.5
>>> 5 // 2
2
>>> 5 % 2
1
```

Python Operators

Operators combine values to give new values.

These values can be taken from variables.

The results can be stored in other variables.

The common operators are:

- \+
- \-
- *
- **/ and // and %**

```
>>> 5 / 2
2.5
>>> 5 // 2
2
>>> 5 % 2
1
```

From the top:

- Division.
- Integer Division ("whole parts").
- Modulus ("remainder")

Python Operators

Operators combine values to give new values.

Precisely what operators achieve depends on the types of the variables.

This is easiest to see from examples, opposite.

```
>>> 2 + 2
4

>>> "Spam! " + "Spam!"
'Spam! Spam!'

>>> 2 * 3
6

>>> "Spam! " * 3
'Spam! Spam! Spam! '
```

Python Operators

Operators combine values to give new values.

Precisely what operators achieve depends on the types of the variables.

This is easiest to see from examples, opposite.

Not every combination works.

It makes no sense to "divide" a string.

```
>>> "Spam! " / 3
Traceback (most recent call last):
  File "<stdin>", line 1, in
<module>
TypeError: unsupported operand
type(s) for /: 'str' and 'int'
```

PyCharm Demo Time

GitHub Code

All the code from the lectures can be found on GitHub, round about here:

https://github.com/TonyJenkins/cfs2160-2018-python-public.git

This will be a record of any code from the lecture, complete examples off slides, and so on.

Java code will appear later in a separate repo with the obvious part of the name changed.

Jobs

Before next week, make sure you have:

- ➢ Got the book, and read the first two chapters.
- ➢ Started to work with PyCharm, using the Python console.
- ➢ Completed a Git tutorial.

And, optionally:

- ➢ Connected PyCharm to your GitHub account.
- ➢ Understood "Public" and "Private" Git repositories.

Jobs

Before next week, make sure you have:

- ➢ Got the book, and read the first two chapters.
- ➢ Started to work with PyCharm, using the Python console.
- ➢ Completed a Git tutorial.

And, optionally:

- ➢ Connected PyCharm to your GitHub ac
- ➢ Understood "Public" and "Private" Git

As before, there is a lot here.

If you are in doubt, or can't work something out - Ask!

Jobs

Before next week, make sure you have:

- ➢ Got the book, and read the first two chapters.
- ➢ Started to work with PyCharm, using the Python console.
- ➢ Completed a Git tutorial.

And, optionally:

- ➢ Connected PyCharm to your GitHub ac
- ➢ Understood "Public" and "Private" Git

From now on, 10 minutes of every lecture are kept for answering (anonymous) questions!

If in doubt - Ask!


python
powered