CSC 108H: Introduction to Computer Programming

Summer 2012

Marek Janicki

Welcome

- Please ask questions/let me know if I'm difficult to understand.
- This is an introduction to computer programming using Python.
  - The order matters!
- Intended for people with no experience with programming.

Course Website

http://www.cs.toronto.edu/~quellan/courses/csc108/

- Note that most of the stuff in the first part of lecture is covered in the info sheet available from the course website.

Is CSC 108H for me?

- CSC 148H is offered during this term.
  - Instructor is Orion Buske.
  - Assumes knowledge of basic python and object oriented concepts.
  - Does more object oriented stuff and focuses on data structures.
  - Lecture is R:4-6, One 2 hour lab per week.
  - http://www.cdf.toronto.edu/~csc148h/summer/

Well, how can I tell?

- CSC 148H is having a one-day ramp-up.
  - Saturday May 19th 10am - 4pm and
    Saturday May 26th 10am - 4pm in BA 3185.
  - http://www.cdf.toronto.edu/~csc148h/summer/rampup.shtml
- Intended for people haven't taken CSC 108H but have done some object-oriented programming.
- I encourage you do show up if you're uncertain which course you should be taking.
  - Please register if you're going.

What will I be doing?

Work	Weight	Comment
Assignments(3)	10%,10%,12%	
Midterm	13%	
Labs(10)	5%	0.5% each.
Exercises(4)	2%, 2%, 3%, 3%	
Final	40%	Need to get at least 40% to pass the course

Assignments!

- They will be posted on the website.
- Due 11:59pm on due date, submitted online.
- You will have the option to partner with one other person for at least two assignments.
- Not required to be monogamous.
- Can use discussion board and labs to meet people.

Late Policy

- You have 2 grace days.
- Each grace day can be used to get a 24 hour extension on an assignment only.
  - You must use grace days in increments of 1.
  - Grace days cannot be stacked, if you wish.
- A team requires two grace days to get an extension.
  - Each partner in a team must contribute one grace day.

Exams!

- A midterm and a final.
- No, I don't know when or where either are yet.
  - When I find out, I will post it on the website and the forum.
  - The midterm will probably be Jun 28th, in the evening.
- They will be closed book written tests.

Labs!

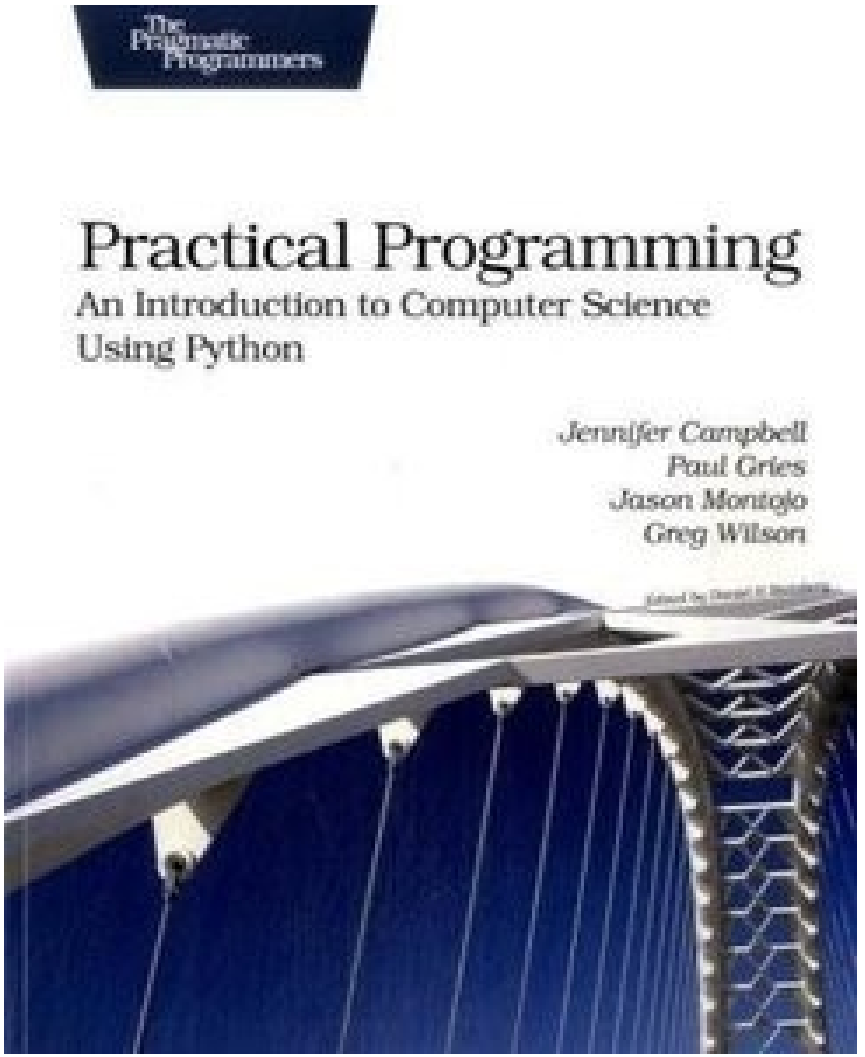
- Labs are done with a partner that is separate from your assignment partner(s).
- They are the tutorials that you sign up for on ROSI.
- They start next week.
- The room assignments are posted on the website.
  - 3 of you have not signed up for a tutorial as of yesterday.

Exercises!

- These are smaller assignments.
- They are only automarked.
- You will be able to submit before the deadline and see the results of the automarking on Markus.
- Will generally have 7~14 days to submit before the deadline.
- No remarks will be given for any reason.

The Book.

- Practical Programming: An Introduction to Computer Science Using Python.
- Can get it cheaply on Amazon.
- Authors from the department.



Getting Help.

- Office Hours.
  - We're deciding on these right now!
- Can ask for help from your TA during labs.
- Course Discussion board.
  - Link on website.
- Undergraduate Help Centre, BA 2270 2-4, Monday-Thursday.
  - Start next Tuesday.

More Help.

- If you can't make office hours or have extenuating circumstances, you can e-mail me.
  - Use quellan@cs.toronto.edu
  - Not quellan@cdf.toronto.edu
  - Please check the discussion board first.
- If you need more practice or another perspective, check the getting help section of the website.

Academic Offences

- You should do all the work that you submit (work by your assignment partner counts).
- Never look at another team's works.
- Never show another team your work.
- Applies to all drafts and partial solutions.
- Discuss how to solve an assignment only with course staff.

Feedback

- You can also give anonymous feedback via the feedback tab on the website.

Administrative stuff that you can do!

- Read the course information sheet.
- Make sure you can find the website and discussion board.
- Buy textbook.
- Look up your CDF username.
  - Need this to submit exercises/do labs!
- If you're working on your own machine, install the software under Python on the course website.

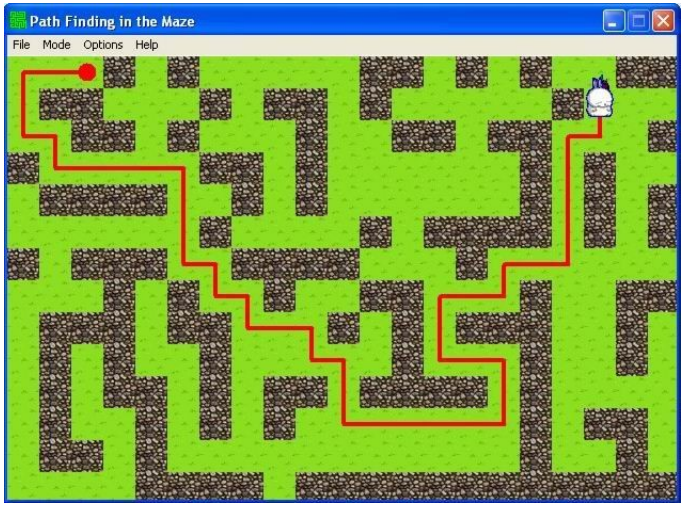
Break, the first.

What is CSC 108H about?

- Learning the basic tools of programming.
  - We use Python for this, but the tools apply to most languages, and even scripts and macros.
- Being able to take human problems, and use programming to solve them.
- Have a better sense of what computer science is about.
  - See how computer science can be applied to climate modelling, bioinformatics, medical science,etc.

Why Programming?

- Powerful and general.
- Can hide a poem in a picture.
- Can remove redeye.
- Allows people to communicate securely.
- Can find optimal paths in huge maps.



What is programming?

- A program is essentially a series of instructions.
  - Like a recipe, or a knitting pattern.
- So why not use English?
  - Too vague and dependent on context.
    - “Eats shoots and leaves”.
  - CPUs have a limited set of instructions.
- We need a language that is unambiguous.

Python!

- Can be translated into a language that the CPU speaks.
  - With no translation errors.
- Python is much more precise than English.
  - Means every detail needs to be specified.
- Python is the language, but what reads it?

Wing

- IDE (Integrated Development Enviroment)
- A set of tools used to help us develop code.
- For now we can think of it as the program that translates our python code for the CPU.
- A free version is linked from the website.

Common Pitfalls

- Not understanding what each line of code is supposed to do.
  - Will cause mistakes if you copy one batch of code from one program to another.
  - Prevents you from being able to effectively write your own code.
- Not being able to trace code.
  - This prevents you from being able to combine multiple lines of code.

Types

- Every base object in python has a type.
- Know what type every object you are using is.
- Useful for sanity checks.

Python as a Calculator

- The shell will interpret lines of python that we feed it.
  - Thus it is useful to check the type of any expression we are using.
  - So we can be sure that we agree with python as to what we are doing.
- Basic mathematical operations are part of python.
  - So we can use python as a calculator.

Python isn't very good at calculating

- You have multiplication, addition, subtraction, division remainder, and powers (*,+,-,/,%,**) but sometimes the answers are weird.
- If you give python integers, it will assume that you want integers back.
- For fractions, one uses floating point numbers.
  - Python interprets any number with a decimal in it as a float.
- Floats are only approximations of real numbers.

Variables

- A variable is a name that refers to a value.
- Variables let us store and reuse values in several places.
- But to do this we need to define the variable, and then tell it to refer to a value.
- We do this using an assignment statement.

Assignment Statements

- Form: `variable = expression`
  - An expression is a legal sentence in python that can be evaluated.
  - So far we've put in math expressions into the shell and seen them be evaluated to single numbers.
- What it does:
  - 1. Evaluate the expression on the RHS.(This value is a memory address)
  - 2. Store the memory address in the variable on the LHS.

Assignment Statements.

- 1. Evaluate the expression on the RHS.(This value is a memory address)
- 2. Store the memory address in the variable on the LHS.

- What this means is that a variable is a name and a memory address. The name points to a memory address where the value is stored.
- This means that variables in python behave fundamentally differently than variables in math.
  - Understanding is required to be able to trace code!

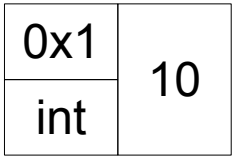
Tracing Code with Variables

- When tracing code, we imagine the variables as names, and their values as objects they refer to.
- We draw names on one side, and the objects they refer to on the other.

Tracing Code with Variables

- When tracing code, we imagine the variables as names, and their values as objects they refer to.
- We draw names on one side, and the objects they refer to on the other.

x: 0x1

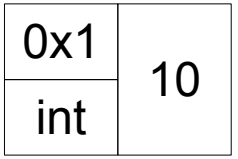


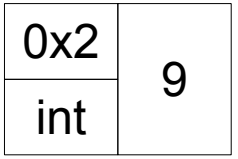
Tracing Code with Variables

```
x = 10
y = 5+4
```

y: 0x2

x: 0x1





Tracing Code with Variables

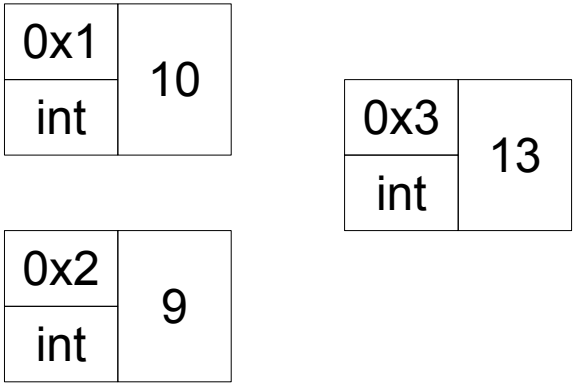
x = 10

y = 5+4

x = 13

y: 0x2

x: 0x3



Tracing Code with Variables

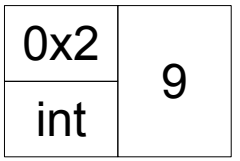
x = 10

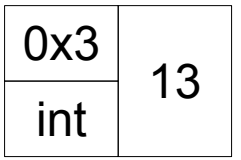
y = 5+4

x = 13

y: 0x2

x: 0x3





Break, the second.

```
x = 15
y = 10
y = x
x = x + 1
y = x + y
```

```
x = 10
y, x = 15
x = x + 1
y = x + y
```

- Which one of the two pieces of code above is legal, and what are the values at the end?

Break, the second.

```
x = 15
y = 10
y = x
x = x + 1
y = x + y
```

```
x = 10
y, x = 15
x = x + 1
y = x + y
```

- Which one of the two pieces of code above is legal, and what are the values at the end?

Break, the second.

```
x = 15
y = 10
y = x
x = x + 1
y = x + y
```

```
x = 10
y, x = 15
x = x + 1
y = x + y
```

- x refers to 16
- y refers to 31

Functions

- Sometimes we want to reuse code, with slightly different variables.
- If we need to take the average of lots of pairs of numbers, we could do

```
x = (num1 + num2)/2
```

- And then everywhere we need an average, we copy this code, and change the variable name.
- But what if there's a mistake?
  - Need to change all the places we take this average.

Functions

- Instead we can reuse code with functions.
- If we have the following somewhere:

```
def avg(num1, num2):
    return (num1 + num2)/2
```

- We can replace x = (num1 + num2)/2 with
  x = avg(num1, num2)
- Now to fix the problem with our average we only need to change the return statement to:

```
return (num1 + num2)/2.0
```

Functions

- A function definition has the form:

```
def function_name(parameters):
    block
```

  - def is a python keyword; it cannot be used for naming functions or variables.
  - A parameter of a function is a variable. A function can have any number of parameters, including 0.
  - A block is a sequence of legal python statments.
    - A block must be indented.
  - If the block contains the keyword return, it returns a value; otherwise it returns the special value None.

Functions

- Defining a function is different from calling it.
- Think about creating a recipe, vs actually cooking it.
- If we create a recipe for a cake, we don't have any cake yet, we only know how to create one.
- But once we have a recipe, we can create as many cakes as we like.

Functions and Variables

- Consider the following Code:

```
def foo(y):
    z = y
    return z
x = 10
foo(x)
print z
```

- What happens?

Functions and Variables

```
def foo(y):
    z = y
    return z
x = 10
foo(x)
print z
```

- What happens?

- Functions can have variables that exist only within the function.
  - These are called local variables.

Functions and Variables

```
def foo(y):
    z = y
    return z
x = 10
foo(x)
print z
```

- What happens?

- Functions can have variables that exist only within the function.
  - These are called local variables.
  - They exist only within the red rectangle.

Functions and Local Variables

- Recall the generic definition of a function:

```
def function_name(parameters):
    block
```

remainder of code

  - Variables defined inside of a function are called local.
    - This includes the parameters.
  - Variables defined outside of a function are called global.

Functions and Local Variables

- Recall the generic definition of a function:

```
def function_name(parameters):
    block
```

  remainder of code

  - Variables defined inside of a function are called local.
    - This includes the parameters.
  - Variables defined outside of a function are called global.
  - Local variables live in the red box.
  - Local variables override global variables with the same name.

Functions and Variables

- Consider the following Code:

```
def foo(x):
    x = 11
    return x
x = 10
print x
print foo(x)
```

- What gets printed?

Functions and Variables

- Consider the following Code:

```
def foo(x):
    x = 11
    return x
x = 10
print x
print foo(x)
```

- What gets printed? 10, then 11. Why?

Functions and Variables

```
def foo(x):
    x = 11
    return x
x = 10
print x
print foo(x)
```

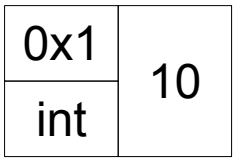
- Let's trace the code.

Functions and Variables

```
def foo(x):
    x = 11
    return x
x = 10
print x
print foo(x)
```

x = 0x1

- Let's trace the code.

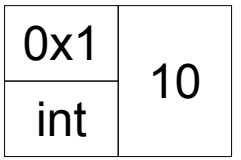


Functions and Variables

```
def foo(x):
    x = 11
    return x
x = 10
print x
print foo(x)
```

x = 0x1

- Let's trace the code.

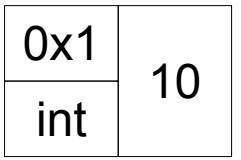


Functions and Variables

```
def foo(x):
    x = 11
    return x
x = 10
print x
print foo(x)
```

x = 0x1

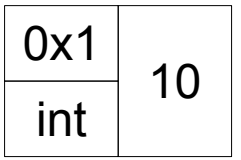
- Let's trace the code.



Functions and Variables

```
def foo(x):
    x = 11
    return x
x = 10
print x
print foo(x)
```

x = 0x1



- Let's trace the code.
  - We need to step into the function.

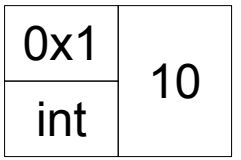
Functions and Variables

```
def foo(x):
    x = 11
    return x
x = 10
print x
print foo(x)
```

x = ?

x = 0x1

- Let's trace the code.

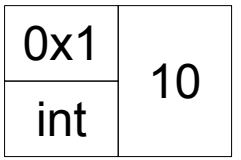


Functions and Variables

```
def foo(x):
    x = 11
    return x
x = 10
print x
print foo(x)
```

x = ?

x = 0x1



- Let's trace the code.
  - Need to differentiate between local and global variables

Functions and Variables

```
def foo(x):
    x = 11
    return x
x = 10
print x
print foo(x)
```

foo locals:	x = ?
Globals:	x = 0x1

0x1	10
int	

- Let's trace the code.
  - Need to differentiate between local and global variables

Functions and Variables

```
def foo(x):
    x = 11
    return x
x = 10
print x
print foo(x)
```

foo locals:	x = ?
Globals:	x = 0x1

0x1	10
int	

- Let's trace the code.
  - Need to evaluate the parameter for foo.
  - foo(x) is in global scope, uses global x.

Functions and Variables

```
def foo(x):
    x = 11
    return x
x = 10
print x
print foo(0x1)
```

foo locals:	x = ?
Globals:	x = 0x1

0x1	10
int	

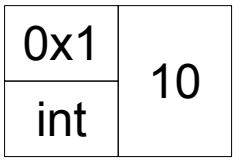
- Let's trace the code.
  - Need to evaluate the parameter for foo.
  - foo(x) is in global scope, uses global x.

Functions and Variables

```
def foo(x):
    x = 11
    return x
x = 10
print x
print foo(0x1)
```

foo locals:	x = 0x1
Globals:	x = 0x1

- Let's trace the code.
  - Now we can assign local value of x.



Functions and Variables

```
def foo(x):
    x = 11
    return x
x = 10
print x
print foo(x)
```

foo locals:	x = 0x1
Globals:	x = 0x1

0x2	11
int	

0x1	10
int	

- Let's trace the code.

Functions and Variables

```
def foo(x):
    x = 11
    return x
x = 10
print x
print foo(x)
```

foo locals:	x = 0x2
Globals:	x = 0x1

0x2	11
int	

0x1	10
int	

- Let's trace the code.
  - To determine which x we choose, we start at the top and move down.

Functions and Variables

```
def foo(x):
    x = 11
    return x
x = 10
print x
print foo(x)
```

foo locals:	x = 0x2
Globals:	x = 0x1

0x2	11
int	

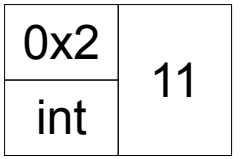
0x1	10
int	

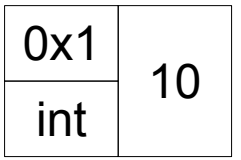
- Let's trace the code.

Functions and Variables

```
def foo(x):
    x = 11
    return 0x2
x = 10
print x
print foo(x)
```

foo locals:	x = 0x2
Globals:	x = 0x1



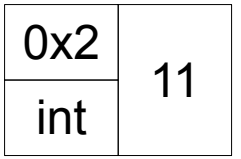


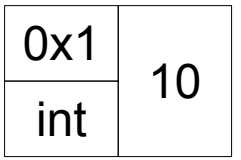
- Let's trace the code.

Functions and Variables

```
def foo(x):
    x = 11
    return 0x2
x = 10
print x
print 0x2
```

Globals:	x = 0x1





- Let's trace the code.
  - When the function is called, we kill local variables, and return the memory address.

Functions and Variables

```
def foo(x):
    x = 11
    return x
x = 10
print x
print 0x2
```

Globals:	x = 0x1

0x2	11
int	

0x1	10
int	

- Let's trace the code.
  - So we can see why the second return value is 11.

Functions and Comments

- Often functions have complicated code.
- To make it easier for humans to understand, we often put in english sentences that we tell the computer to ignore.
  - These are called comments.
- Two ways of commenting in python:

```
#The computer ignores this line.

'''The computer ignores all the lines between triple quotes, regardless of how many there are.'''
```

Functions and Types

- Recall that every base object in python has a type.
- For now, it is useful to think of functions as things that take base objects of some types and generate new base objects that have types.
- So it is a recipe that takes some base objects and produces a new base object.

Function Conventions

- Recall the format of a function:

```
def function_name(parameters):
    block
```

- This is all that is legally required for a function, but in practice we really use:

```
def function_name(parameters):
    '''(parameter types)-> output type
    Description of what the function does.'''
    block
```

Function Conventions

```
def avg(num1, num2):
    return (num1 + num2)/2.0
```

- Should actually be:

```
def avg(num1, num2):
    '''(int/float, int/float) -> float
    Takes two numbers and returns their average.'''
    return (num1 + num2)/2
```

Function Conventions

```
def avg(num1, num2):
    '''(int/float, int/float) -> float
    Takes two numbers and returns their average.'''
    return (num1 + num2)/2
```

- Not:

```
def avg(num1, num2):
    '''(int/float, int/float) -> float
    Takes two numbers and returns their average
    by adding them and dividing the result by 2.0.'''
    return (num1 + num2)/2
```

Function Conventions

```
def avg(num1, num2):
    '''(int/float, int/float) -> float
    Takes two numbers and returns their average.'''
    return (num1 + num2)/2
```

- Not:

```
def avg(num1, num2):
    '''(int/float, int/float) -> float
    Takes two numbers and returns their average
    by adding them and dividing the result by 2.0.'''
    return (num1 + num2)/2
```

Naming Conventions.

- Naming rules and conventions apply to functions, variables and any other kind of name that you will see.
- Must start with a letter or underscore.
- Can include letters, numbers, and underscores and nothing else.
- Case matters, so age is not same name as Age.

Naming Conventions.

- Python Convention: pothole_case
  - That is, all lower case, and underscores seperate words.
- CamelCase is sometimes seen, but not for functions and variables.
  - That is, capital letters separate words.
- Single letters are rarely capitalised.
- These conventions are important for legibility which factors into maintaining code.

Python comes with a lot of stuff.

- We saw how to write our own functions, but python comes with lots of prebuilt functions in Python.
- Some math ones like max and abs.
- But also other useful ones like dir and help
  - dir returns a list of functions that are available.
  - help returns information about a function or module.

Types

- Every Python value has a type that describes what sort of value it is and how it behaves.
- There is a built in function `type` that returns the type of an expression.
  - Useful for sanity checks so that you are sure that you and python agree as to what your line of code is doing.
  - Can use it to check the type of a variable, and of a function call.

Type is more useful than the shell.

- Consider the following two functions:

```
def foo(x):
    return x
```

```
def goo(x):
    print x
```

- foo(9) and goo(9) look the same in the shell.
- But type(foo(9)) and type(goo(9)) highlights the fact that the two functions behave differently.

Home Stretch

- To finish off, we'll see how to create a non-trivial program quite quickly.
  - Some of the stuff we'll be using is a bit advanced, so don't worry if you don't completely follow everything.
- A lot of people create external modules that extend the capabilities of python.
  - We'll be using the media module, which was created by UofT students.
  - To use a module we import it with import module_name

Media Module

- The basic function of the Media Module is to show pictures.
  - pic = media.load_picture(filename) loads an image into pic.
  - media.show(pic) shows the picture.
- We want to use this to design a program that can take a picture, and make it appear as if it was taken at sunset.

How do we do that?

- Well, we take what we know about image files.
- Basically we know that images files are really many tiny coloured squares called pixels.
- Since we have RGB monitors, this means each colour is a combination of red, green and blue.
- It turns out that the pixel colours are specified by 3 numbers between 0 and 255 that say how much red green and blue each pixel has.
  - So (255,0,0) is red, while (0,255,0) is green and so on.

Leveraging our Knowledge.

- So we know about pixels.
- What do we know about sunset?
  - Colours tend to be redder and less blue or green.
- So if we could change the colour values of each pixel accordingly, we'd probably do pretty well.
  - So let's try decreasing blue and green by 70%,

Pseudo-Code version.

- We want something like:
- For every pixel,

  get the (blue/green) component of that pixel.

  Reduce this component by 30%

  set the (blue/green) component of that pixel to the new value.
- We're in luck, as there's a way to quickly go over all the pixels.

A General Approach

- While admittedly all planned beforehand, the way we approached the problems was in three stages.
  - Design: We thought about what the right approach was before writing any code.
  - Code: Once we thought we had a good idea, we wrote the code.
  - Verify: we tested our code to make sure we weren't making any dumb mistakes.