JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

EN.540.635
Software Carpentry

Lecture 4
More Python Basics and the Python Imaging Library

What We've Covered So Far…


```python
import math


def cartesian_to_polar(x, y):
    # NOTE! This code only works for x > 0 and y > 0

    # If x == y == 0, this is a weird boundary case we accomodate for
    if x == y and y == 0:
        return 0, 0

    # Get the polar radial distance
    r = math.sqrt(x ** 2 + y ** 2)
    # Get the polar angle
    # NOTE! We need to accomodate for x = 0, as we will be dividing
    if x == 0:
        # It should be either positive or negative infinity, depending on
        # the sign of y
        z = float('inf') * y / abs(y)
    else:
        z = float(y) / float(x)
    theta = math.atan(z)
    # Convert angle to degrees, as default math.atan is radians
    theta *= 180.0 / math.pi
    # Now, we only have the angle
    return r, theta


# Define our x and y coordinates
x, y = 0, 1
# Call our conversion function
r, t = cartesian_to_polar(x, y)
# Print the result
print("Converted (%.2f, %.2f) to (r, theta) = (%.2f, %.2f)" % (x, y, r, t))
```

- Variables
- Conditionals
- Loops
- Functions

- To be covered:
  - Code readability
  - PEP 8
  - Python Imaging Library

Code Readability

- It is important that your code runs as intended, but it is also important that your code can be easily understood by others.

- Important things that can improve the readability of your code:
  - Good variable/function names
  - Good code organization
  - Comments and docstrings
  - Proper code formatting

Variable and Function Names

- Variable names should be as short as possible while still properly conveying what data the variable holds.
- Avoid using indistinguishable characters as single-character variable names (“l”, “O”, “I”).
- It is common practice to use all lowercase characters, with words separated by underscores for both variable and function names.
- The example code on slide 2 is a good example of proper naming.

Python Keywords

Python has a list of keywords that are commonly used in code.

These keywords cannot be used as variable or function names:

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

Code Organization

"I have always felt that programming is telling a computer a story. If your story is good, the computer will execute it efficiently, if the story sucks, the execution sucks. I write code like I would write a technical paper, it has direction and flow, and you can tell when you have reached the conclusion of the story."

- from Dan Bader's Python newsletter

Simple code organization strategy for a Python script:

1. Import statements
2. Functions
3. The main code you want to run

(again, the example code on Slide 2 is a good example).

Comments

- Explain what your code is doing!

- For yourself:
  - If you have to go back to code you've written a long time ago, it is good to have good comments so you can easily remember what your code does.

- For others:
  - Writing code can be a collaborative experience.
  - It is important that other people can easily understand your code.

- Rule of thumb: use good variable and function names, and then add comments when necessary.

Docstrings

- Documentation strings (docstrings) are comments for functions, with a specific format:
  - What does the function do?
  - What values need to be input to the function?
  - What are the output values of the function?

```
1  import math
2
3
4  math.sqrt
       sqrt        function    math.sqrt Return the square root of x.
```

- Simple functions (1 or 2 lines) only need a quick description. Longer functions should have a description, along with the required inputs and outputs.

Docstrings

Description

```
def grayscale(fname, method=0, ext=".png"):
    '''
    This function will save an image as grayscale.

    **Parameters**

        fname: *str*
            The file name, without a file extension.
        method: *int*
            Which method to use:
                0 - Average
                1 - RMS
        ext: *str*
            The extension (ex. .png) that we read and write in.

    **Returns**

        None
    '''
```

Input Parameters

Output Values

PEP 8 Styling

- PEP 8 is the agreed-upon style guide for how to format Python code.
- The main purpose of a style guide is to have good readability and consistency.
- A lot of rules regarding:
  - Indentation
  - Blank lines
  - Whitespace
  - Comments
  - Naming Conventions
- Pretty much everything discussed in the previous slides is part of PEP 8 styling.

PEP 8 Styling

- There are a lot of rules to remember, and it can be annoying to have to worry about formatting things when you are writing code.
- Linters – tools that analyze code and flag for programming errors and formatting issues.
- Many text editors geared towards programming have linters built into them.

Linters in Sublime

If you install the Anaconda package in Sublime Text, it comes with a linter and a tool to help with automatic formatting.

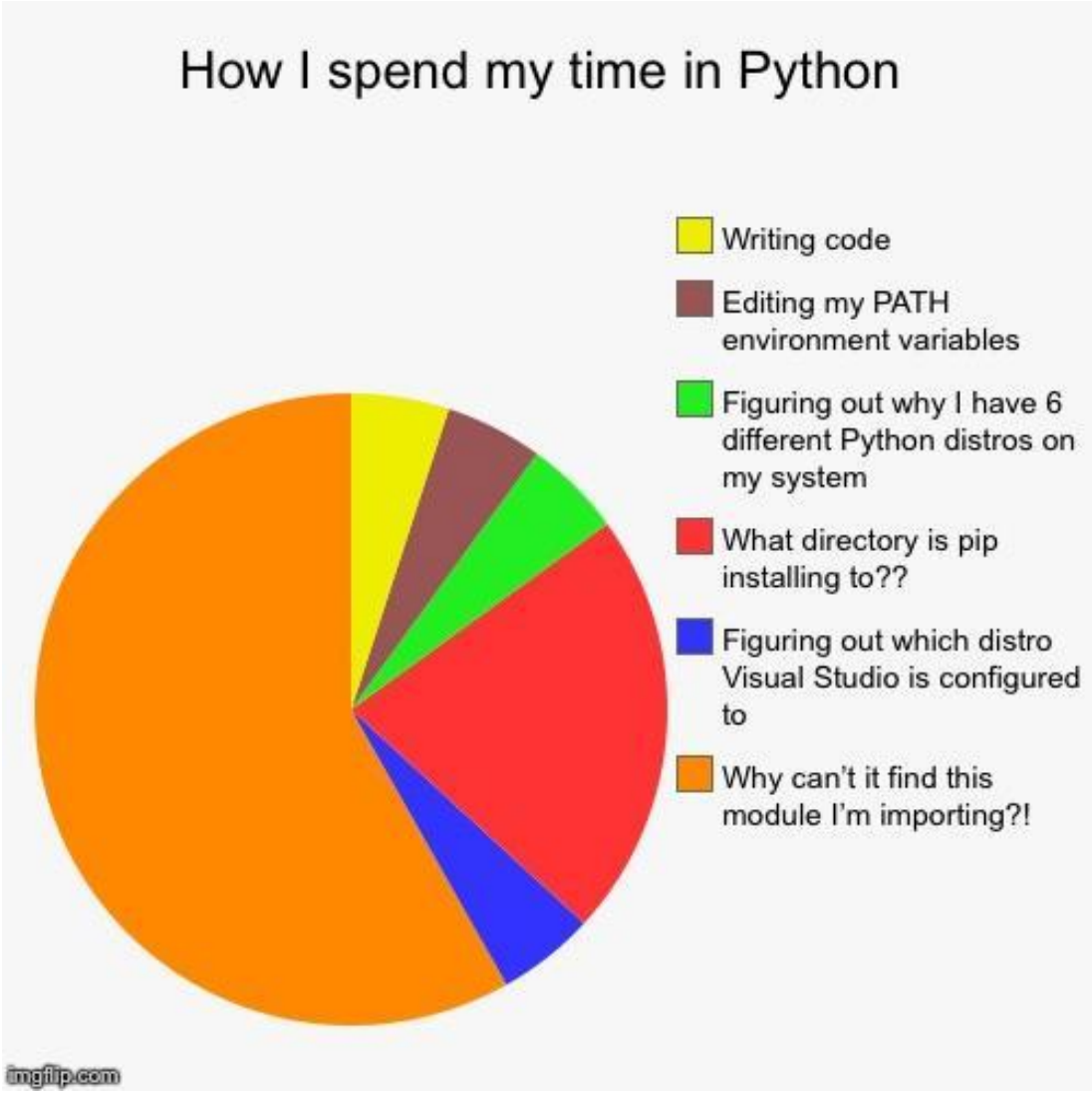
```
6   def cartesian_to_polar(x, y):
7       # NOTE! This code only works for x > 0 and y > 0
8
9       # If x == y == 0, this is a weird boundary                    date for
10      if x == y and y == 0:
11          return 0, 0
12
13      # Get the polar radial distance
14      r = math.sqrt(x ** 2 + y ** 2)
15      # Get the polar angle
16      # NOTE! We need to accomodate for x = 0, as we will be di
17      if x == 0:
18          # It should be either positive or negative infinity,
19          # the sign of y
20          z = float('inf') * y / abs(y)
21      else:
22          z = float(y) / float(x)
23      theta = math.atan(z)
24      # Convert angle to degrees, as default math.atan is radia
25      theta *= 180.0 / math.pi
26      # Now, we only have the angle
27      return r, theta
```

Show Diff Hunk
Show Unsaved Changes...
Cut
Copy
Paste
Select All
Open Containing Folder...
Copy File Path
Reveal in Side Bar
Anaconda

Goto Definition
Goto Assignment
Find Usages
Show Documentation
Rename object under cursor
Show error list
Next lint error
Previous lint error
Autoformat PEP8 Errors
McCabe complexity check
Auto import undefined word under cursor
Run tests on current file
Run full project tests= suite
Run test under the cursor
Repeat last test run

Python Modules

- Python has many existing libraries that are available for us to use (especially if we've downloaded Anaconda).
- A module is a file with Python definitions and statements that we can use for our own code.
- Some examples include math, random, PIL, …



Python Imaging Library

- This library is useful for image processing and manipulation using Python. It contains several different modules, each with its own functions.

- The module we will be focusing on is the Image Module:
  - Opening images
  - Manipulating images (at the pixel level)
  - Creating new images
  - Saving images

Python Imaging Library: Opening an Image

PIL.Image.open(*fp, mode='r'*)

Opens and identifies the given image file.

This is a lazy operation; this function identifies the file, but the file remains open and the actual image data is not read from the file until you try to process the data (or call the `load()` method). See `new()`.

Parameters:

- **fp** – A filename (string), pathlib.Path object or a file object. The file object must implement `read()`, `seek()`, and `tell()` methods, and be opened in binary mode.
- **mode** – The mode. If given, this argument must be "r".

Returns: An `Image` object.

Raises: **IOError** – If the file cannot be found, or the image cannot be opened and identified.

Python Imaging Library: Pixel Manipulation

`Image.getpixel`(*xy*)

Returns the pixel value at a given position.

Parameters: **xy** – The coordinate, given as (x, y).

Returns: The pixel value. If the image is a multi-layer image, this method returns a tuple.

`Image.putpixel`(*xy, value*)

Modifies the pixel at the given position. The color is given as a single numerical value for single-band images, and a tuple for multi-band images.

Note that this method is relatively slow. For more extensive changes, use `paste()` or the `ImageDraw` module instead.

See:

- `paste()`
- `putdata()`
- `ImageDraw`

Parameters:
- **xy** – The pixel coordinate, given as (x, y).
- **value** – The pixel value.

Python Imaging Library: Creating a New Image

`PIL.Image.new`(*mode, size, color=0*)

Creates a new image with the given mode and size.

Parameters:

- **mode** – The mode to use for the new image. See: Modes.
- **size** – A 2-tuple, containing (width, height) in pixels.
- **color** – What color to use for the image. Default is black. If given, this should be a single integer or floating point value for single-band modes, and a tuple for multi-band modes (one value per band). When creating RGB images, you can also use color strings as supported by the ImageColor module. If the color is None, the image is not initialised.

Returns: An `Image` object.

Python Imaging Library: Saving an Image

`Image.save`(*fp, format=None, **params*)

Saves this image under the given filename. If no format is specified, the format to use is determined from the filename extension, if possible.

Keyword options can be used to provide additional instructions to the writer. If a writer doesn't recognise an option, it is silently ignored. The available options are described in the *image format documentation* for each writer.

You can use a file object instead of a filename. In this case, you must always specify the format. The file object must implement the `seek`, `tell`, and `write` methods, and be opened in binary mode.

Parameters:

- **fp** – A filename (string), pathlib.Path object or file object.
- **format** – Optional format override. If omitted, the format to use is determined from the filename extension. If a file object was used instead of a filename, this parameter should always be used.
- **options** – Extra parameters to the image writer.

Returns: None

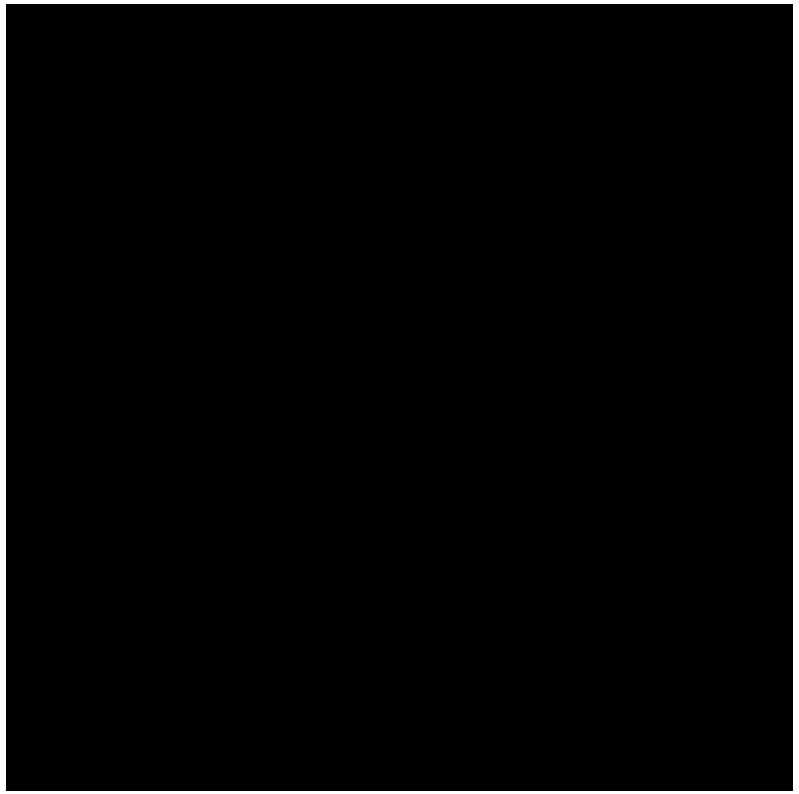
Raises:

- **KeyError** – If the output format could not be determined from the file name. Use the format option to solve this.
- **IOError** – If the file could not be written. The file may have been created, and may contain partial data.

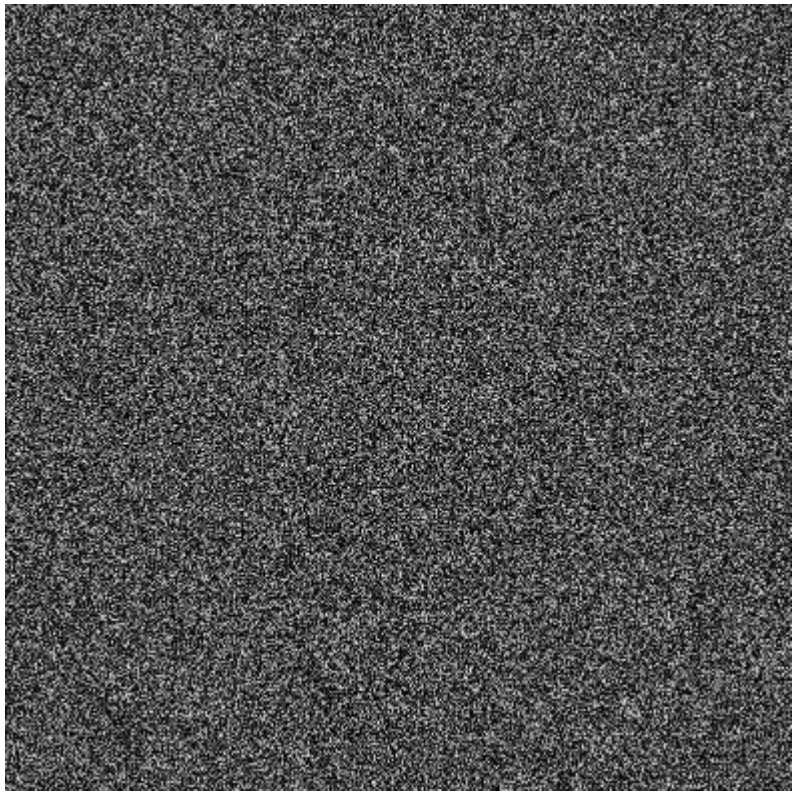
Basic Example

```
import random
from PIL import Image

# Make a completely black image, 1000x1000 pixels
SIZE_W, SIZE_H = 1000, 1000
img = Image.new("RGB", (SIZE_W, SIZE_H), color=(0, 0, 0))
img.save("all_black.png")

# Open this image back up for further processing.
# We will just add in random, gray-sacle noise
img2 = Image.open("all_black.png")
width, height = img2.size
N_PIXELS_TO_CHANGE = 1000000
for i in range(N_PIXELS_TO_CHANGE):
    x = int(random.random() * SIZE_W)
    y = int(random.random() * SIZE_H)
    c = int(random.random() * 255)
    img2.putpixel((x, y), (c, c, c))
img2.save("static.png")
```

Basic Example



all_black.png



static.png