Computer Science 1 — CSci 1100
Lecture 5 — Modules, Objects, Methods, Images

Reading

This material is drawn from Chapters 4 of *Practical Programming*, but we will not use the `media` module discussed there, opting instead for the more widely-used `PIL` (Python Imaging Library) and the modules in PIL.

Overview

- Modules are a way of providing a set of functions and even constants to be used in other programs.
- All variables are Python "objects".
- New types of objects bring new functionality with modules specifically designed for these objects.
- Our aim in this lecture is to learn how we can use these modules and define our own modules.

Modules

- Python modules combine functions and constants.
- They allow functions to be defined once, used many times and in different programs.
- Before we can use a module, we need to import it.

```
>>> import module_name
```

- Importing makes the functions in the module available for use.

Using Existing Modules

- Python has many useful built-in modules. An important example is the `math` module:

```
>>> import math
>>> math.sqrt(5)
2.2360679774997898
>>> math.trunc(4.5)
4
>>> math.ceil(4.5)
5.0
>>> math.log(1024,2)
10.0
>>> math.pi
3.1415926535897931
```

- Note: all the main functionality of Python is part of the `__builtins__` module. You can see its contents by typing:

  ```
  >>> help(__builtins__)
  ```

We can also write our own modules

- Any script we have written can be imported as a module.
- Let us write a simple module named `name_print` module.
- A module will have a corresponding .py file, so `name_print` module can be found in `name_print.py`.

  ```
  """ A set of name printing methods

      first_name middle_initial last_name
      last_name, first_name middle_initial
      assumes there is a dot after the middle initial
      and middle initial exists

  """

  def full_name(first, middle, last):
      """Return the full name in the form: first middle last"""

      return "%s %s %s" %(first, middle, last)

  def full_reversed(first, middle, last):
      """Return the full name in the form: last, first middle"""

      return "%s, %s %s" %(last, first, middle)

  spam_name = 'Rick Astley'

  print spam_name
  ```

Importing multiple times

- What happens when we import a module many times?

  ```
  >>> import name_print
  >>> import name_print
  ```

- All the code in the module is executed once when we import it first. All functions and variable values are stored in the shell.
- The second time we try import, Python will just skip it because it is already imported.
- If a module is changed, you must force Python to reload the module.

Importing modules

- `import name_print` moves all functions in the module under a namespace called `name_print`

```
>>> import name_print

>>> name_print.full_name('James','T.','Kirk')
>>> name_print.spam_name
```

- You can find out what is in a module by typing help. Note that the documentation strings become very handy.

```
>>> help(name_print)
```

- Sometimes typing `name_print.` is annoying, especially if we want to use its functions repeatedly. Python offers us a short-hand using the `from` and `import` combination:

```
>>> from name_print import full_name
>>> full_name('James','T.','Kirk')
```

- This imports chosen functions of the module into the root namespace of the Python shell, which means that you do not have to write the name of module. But, this is not generally recommended.
- Try these now:

```
>>> name_print.full_name('James','T.','Kirk')
>>> name_print.full_reversed('James','T.','Kirk')
```

Module Attributes

- Once you import a module, you can see its attributes by typing:

```
>>> help(name_print)
```

- Attributes include the following:
- all the module's functions: `full_name`, `full_reverse`
- all the module's constants: `spam_name`
- built-in attributes that all Python modules have, in particular:
  - `__doc__` is the documentation string
  - `__file__` is the name of the file containing the module
  - `__name__` is the name of the module.
- You can see the value of an attribute, by typing

```
>>> name_print.spam_name
>>> name_print.full_name
>>> name_print.__name__
```

Exercise

- How would you modify the functions from Lecture 3 to make use of the math module?

```
def area_circle(radius):
    pi = 3.14159
    return pi * radius ** 2

def volume_cylinder(radius,height):
    pi = 3.14159
    area = area_circle(radius)
    return area * height

def area_cylinder(radius,height):
    circle_area = area_circle(radius)
    height_area = 2 * radius * pi * height
    return 2*circle_area + height_area

def area_and_volume(radius, height):
    print "For a cylinder with radius", radius, "and height", height
    print "The surface area is ", area_cylinder(radius,height)
    print "The volume is ", volume_cylinder(height, radius)
```

- Given the above program, what is the output of the following?

```
>>> import area_volume

>>> area_volume.area_circle(20)

>>> math.log(100,2)
```

Objects and Methods

- All the variables in Python are objects.
- So far, we have seen three main types of objects: int, float and string.
- We will see many more, starting today!
- Objects are abstractions. They store specific types of data.
- Plus, objects have operations specific to them. We call them **methods**.
- Some methods have the same name, but act differently depending on the type of object.
- Integer division:

```
>>> x = 7
>>> y = 4
>>> x / y
1
```

- Float division:

```
>>> x = 7.0
>>> y = 4
>>> x / y
1.75
```

- The division method for integers is different than the one for floats.

Running Methods for a Specific Object

- Actually, the methods typically apply to a specific object.
- To see what methods apply to an object, you can type:

```
>>> help(int)
>>> help(float)
>>> import string
>>> help(string)
```

- We can also write integer/float division we have just seen as:

```
>>> x.__div__(y)
1.75
```

  which means, apply division method of object type integer/float to x treating it as the numerator, with the additional argument y as the denominator.

- Remember, methods of the form `__method__` are internal to Python. So, `x/y` is a shorthand for `x.__div__(y)`.
- Careful: integer literals are treated as special: `7.__div__(4)` does not work.
- The format for employing methods for a class of objects is
  - Variable name or literal (except for integers)
  - Dot `.`
  - Function name
  - Parentheses, perhaps with arguments

String Methods

- In addition to the length function we have seen, Python provides a set of sophisticated functions for strings.
- Here are some examples

```
>>> s = "GeorGe WasHINgton"
>>> s.lower()
'george washington'
>>> s.upper()
'GEORGE WASHINGTON'
>>> s.capitalize()
'George washington'
```

- You can apply them directly on strings too.

```
>>> 'No Jobs, no Cash, no Hope, ...'.replace('no','some')
'No Jobs, some Cash, some Hope, ...'
```

- Here is another example, converting to small letters and then replacing the first name:

```
>>> s1 = s.lower();
>>> m = s1.replace('george', 'martha')
>>> print m
martha washington
```

Exercise

- Given s:

  `s = 'I may have made a terrible mistake'`

  write code to convert `s` to

  `'I MADE HAVE may A mistake TERRIBLE'`

  by using only the functions below. You can only apply the functions to variable s, but you do not have to use all the functions.

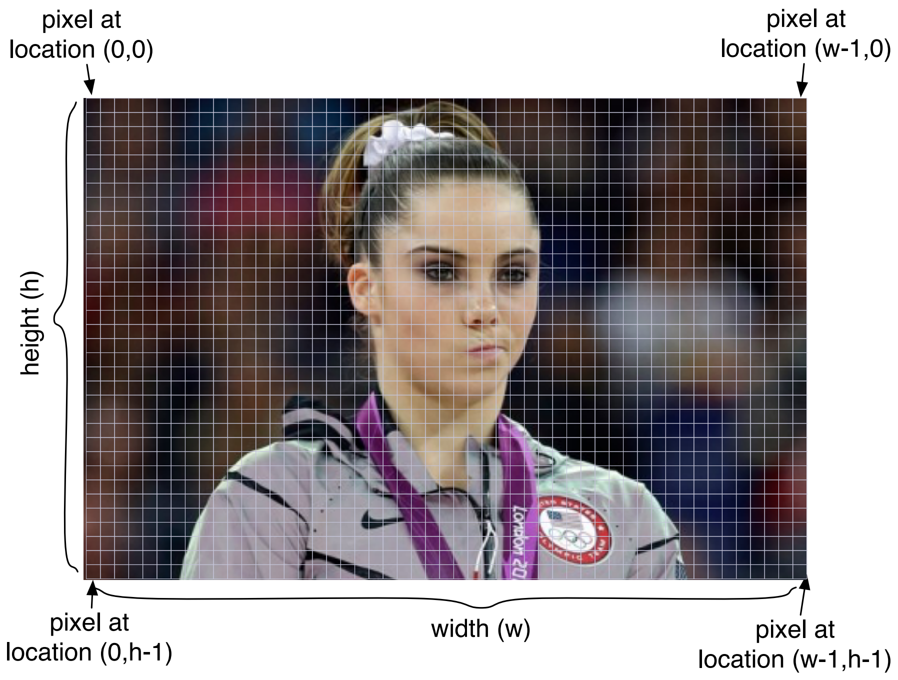
  - `string.upper()`
  - `string.replace(old, new)` replaces all occurrences of string `old` with `new` in string

PIL — Python Image Library

- PIL is a series of modules built around the `Image` type, our first type that is not part of the main Python language.
  - The distinction is not really clear at this point, but we will come back to it eventually.
- We will use images as a continuing example of what can be done in programming beyond numbers and beyond text.
  - The core techniques of Python and, more broadly, computer science apply to all of these.
- Besides, it's fun!

Digging in a Bit: Image Coordinates

- An image is a two-dimensional matrix of pixel values
- For simplicity, we will solely deal with JPG images
- The origin is in the upper left corner



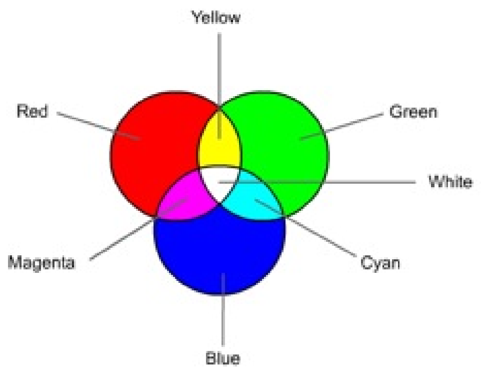
pixel at location (0,0)
pixel at location (w-1,0)
height (h)
pixel at location (0,h-1)
width (w)
pixel at location (w-1,h-1)

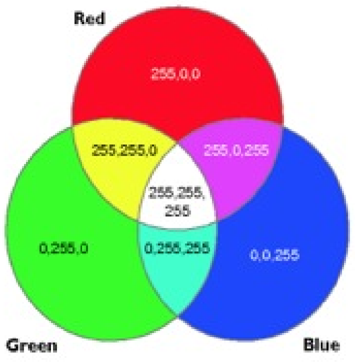
Digging in a Bit: Image Values

- Image mode describes what is stored in each pixel:
  - `'JPG'` images use RGB values — a "3-tuple" of integers representing the red, green and blue intensities, usually in the range 0 to 255
  - `'PNG'` images use RGBA values — a "4-tuple" of integers, where the first three represents the RGB values as before, and A is the alpha channel which represents which pixels should be transparent. This is used for images with transparent backgrounds and is especially useful for pasting images on top of each other.
  - `'L'` means a single value representing a gray scale image.
  - Many other formats (called image mode), but these are the most important, especially to us.

- Some basic colors:

Color	(red,green,blue) value
Black	(0,0,0)
Red	(255,0,0)
Green	(0,255,0)
Blue	(0,0,255)
White	(255,255,255)
Light Gray	(122,122,122)





Some important image modules

- **Image** module contains main functions to manipulate images: open, save, resize, crop, paste, create new images, change pixels, etc.
- **ImageDraw** module contains functions to touch up images by adding text to it, draw ellipses, rectangles, etc.
- **ImageFont** contains functions to create images of text for a specific font.
- We'll see the usage of each in the next examples. Details that we will cover:
  - import
  - opening an image
  - accessing the information about image formats
  - resizing and its argument
  - show
  - save
- Please copy and run these example programs on your virtual machine. Change settings and see what happens.

Image Type and Methods

- Let us now see some very useful image methods
- `im = Image.open(filename)` reads an image with the given filename and returns an image object. It assumes the file is stored in the same directory as your script.
- `im.show()` displays the image
- `im.save(filename)` saves the image to the given file
- Images have the following constants: `im.format, im.size, im.mode`
- `im.resize((width,height))` resizes an image to a given size.
- `im.convert(mode)` changes the mode of an image.

- `im.crop((w1,h1,w2,h2))` creates a new image by cropping the given box from the current image
- `im.transpose(flip_position)` creates the mirror reflection of the image in the given direction

Example: Reading and Displaying

- Here is our first example, stored in file `image_ex1.py`

```
""" Example that shows how to: scale an image,
    convert it to gray scale and save it

"""

import Image

filename = "amy_sheldon.jpg"
im = Image.open(filename)
print '\n', '********************'
print "Here's the information about", filename
print im.format, im.size, im.mode

gray_im = im.convert('L')

scaled = gray_im.resize((128,128))
print "After scaling and converting to gray scale,\
 the information about the image becomes"
print scaled.format, scaled.size, scaled.mode

scaled.show()
scaled.save(filename + "_scaled.jpg")
```

- It is important to note that the `show` function is a bit weak, only allowing display of one image at a time. Primarily it is meant for debugging purposes.

Example: Pasting images on top of each other

- We are going to paste images on top of another.
- Important method to use: `Image.paste(foreground_image,box, mask)` where
  - `foreground_image` is the pasted image
  - `box` is a 4-tuple showing where image should be pasted `(upper_left_x, upper_left_y, lower_right_x,lower_right_y)`.
  - `mask` is an optional argument, telling which pixels from the foreground image should be treated as transparent. This is best obtained from the alpha channel of a PNG image.
- Example usage (no mask):

```
b_image.paste(f_image, (0,0,f_image.width, f_image.height))
```

Example: Cut and pasting parts of an image

- This example is stored in `image-cutandpaste.py`. It crops two boxes from an image, and pastes them at different locations.

```
""" Shows how to crop and paste from an image """

import Image

im = Image.open("amy_sheldon.jpg")

# Set the upper left corners of the Amy and Sheldon regions
# Set a fixed height and width
amy_x = 110
amy_y = 105
sheldon_x = 445
sheldon_y = 40
w = 100
h = 100

# Crop out and reflect Amy and Sheldon's heads
sheldon_head = im.crop((sheldon_x, sheldon_y, sheldon_x+w, sheldon_y+h))
sheldon_reflected = sheldon_head.transpose(Image.FLIP_LEFT_RIGHT)
amy_head = im.crop((amy_x,amy_y, amy_x+w, amy_y+h))
amy_reflected = amy_head.transpose(Image.FLIP_LEFT_RIGHT)

# Paste the regions back into the image and show it
im.paste(sheldon_reflected, (amy_x,amy_y))
im.paste(amy_reflected, (sheldon_x,sheldon_y))
im.show()
```

Summary

- Modules contain a set of functions, constants and even code.
- Functions in a module can be executed by a call of the form: `module_name.function_name(arguments)`
- They can be imported, in which case the functions and constants in them can be used by other modules.
- Module attributes can be viewed by the `dir` function.
- All variables in Python are objects of a specific type.
- Objects have associated functions called methods that apply an operation to an object: `object.method(arguments)`.

Additional Example: Writing text on image

- This example is stored in file `text_on_image.py`
- To be able to add text, you need get an object of type `draw` that you can draw on first.

  ```
  draw = ImageDraw.Draw(im)
  ```

- To draw text, you need to specify its starting location ((100,10)), a font using the font files on your computer with extension .ttf and a size (45).

  ```
  ImageFont.truetype('/usr/share/fonts/truetype/freefont/FreeMonoBold.ttf', 45)
  ```

- Here is the program:

  ```
  """Adds text with a given font to an image"""

  import Image
  import ImageDraw
  import ImageFont

  im = Image.open('ermahgerd.jpg')
  w,h = im.size
  draw = ImageDraw.Draw(im)
  font_path = '/usr/share/fonts/truetype/freefont/FreeMonoBold.ttf'
  draw.text((100, 10), 'EHMARGERD', \
              font=ImageFont.truetype(font_path, 45), \
              fill= (255,255,255))
  draw.text((100, h-80), 'EMEGEES', \
              font=ImageFont.truetype(font_path, 45), \
              fill= (255,255,255))
  im.save('ermahgerd_wtext.jpg')
  ```

- Note: you need to point to the correct path the font lies in.
- Also note: “Impact” is the font used in memes, but it is unavailable in the virtual machine.

Additional Example: Pasting image with a transparent background

- We will use a PNG image with RGBA mode where A is the transparent pixels.
- We will use a mask to only show the non-transparent pixels of the pasted image.
- We will also scale the pasted image first to fit in the background image.
- This example is stored in `image_paste_ex.py`.

```
""" Scales/copies the foreground image to the lower right corner
    of the background image

"""

import Image

foreground_im = Image.open("mm_nobackground.png") #image of McKayla Maroney
background_im = Image.open("jeopardy.jpg") #image of RPI achievement

fwidth, fheight = foreground_im.size
bwidth, bheight = background_im.size

#upper left coordinate where the foreground image will be pasted
upleft_x = int((5.0/6) * bwidth)
upleft_y = int(0.3 * bheight)

#the required size of the pasted image
#assuming the image is taller than wider
paste_height = bheight - upleft_y
paste_width = paste_height * fwidth/fheight

foreground_im_scaled = foreground_im.resize((paste_width, paste_height))

r,g,b,alpha = foreground_im_scaled.split()

background_im.paste(foreground_im_scaled,
              (upleft_x,upleft_y,upleft_x+paste_width,upleft_y+paste_height),
                    mask=alpha)

background_im.save("mm_notimpressed.jpg")
```