Programming: Part II

In this section of notes you will learn about more advanced programming concepts such as looping, functions.

Repetition

- Problem: what if a program or portion of a program needs to repeat itself.
- Example:
  - Allowing a player to re-play a game again.
  - Continuing to prompt a person to type in value if they enter in an invalid response.
- Loops: Allows a program or portion of a program to be repeated
  - There are two main types of loops in Python (for and while).
  - In this course we'll focus on the latter.

The For Loop

Format:

for <*variable*> in <*something that can be stepped through*>:
    body

- **Example:** Available online and is called "loop1.py"

```
def loop1 ():
    total = 0
    for i in range (1, 4, 1):
      total = total + i
      print "i=", i, " total=", total
    print "Done!"
```


Additional For-Loop Examples

- Available online and is called "loop2.py"

```
def loop2 ():
   for i in range (5, 0, -2):
     print "i=", i
   print "Done!"
```

- Available online and is called "loop3.py"

```
def loop3 ():
    for i in [5, 2, 3, 10]:
        print i
    print "Done!"
```

Real: Problem Solving Approaches

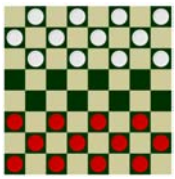
- Bottom up
- Top down

Bottom Up Approach To Design

•Start implementing all details of a solution without first developing a structure or a plan.

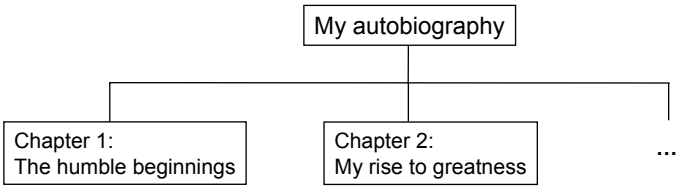
Here is the first of my many witty anecdotes, it took place in a “Tim Horton’s” in Balzac..

−Potential problems:
- −(Generic problems): Redundancies and lack of coherence between sections.
- −(Programming specific problem): Trying to implement all the details of large problem all at once may prove to be overwhelming.



Top Down Design

1. Start by outlining the major parts (structure)



2. Then implement the solution for each part

Chapter 1: The humble beginnings

It all started seven and one score years ago with a log-shaped work station…

Writing Programs Using The Top Down Approach

- Functions can be used to break a program down into manageable parts.
- Each part of the program is defined by a function.
- Each function is written one at a time and tested before being added to the main program.

A Very Simple Example Of Using Functions

Available online and is called "fun1.py"

```
def display ():
    print "Inside display"

def main ():
    print "Starting main"
    display ()
    print "Display is done, back in main"
```

4. Statements inside of display execute one-at-a-time.

2. First statement inside main runs

3. Second statement invokes the function called "display"

5. Execution returns back to "main" and the third and final statement here executes.

1. Starting the main program (function called "main")

```
>>> main()
Starting main
Inside fun
Fun is done, back in main
>>>
```


Issue: What's Inside A Function Stays Inside A Function

- Note: This example won't translate into binary so it can't be run.

```
def fun ():
    print num

def main ():
    num = 10
    print num
    fun ()
```

Num??? Never heard of it!!!

Passing Information Into Functions

- Parameters/Inputs: Allows the contents of a variable to be passed into a function.
- (Modified version of the previous example and it will work):

```
def fun (num):
    print num

def main ():
    num = 10
    print num
    fun (num)
```

The contents of variable 'num' in main are stored in a local variable called 'num' – it's num in fun :D

Passes the contents of 'num' from the main function into function fun

Passing Multiple Parameters/Inputs

Format:

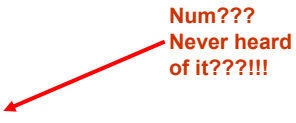
def *function name* (*input*[1], *input*[2],...*input*[n])
    body

Example:

```
def fun (num1, num2, num3, string1)
    num1 = num2 + num3
    string1 = "dude!"
```

In A Similar Fashion Values Must Be Returned From Functions

```
def fun ():
    num = 1
    print num

def start ():
    fun ()
    print num
```



Returning Values From Functions

Format (Writing the function being called):

def *function name* ():
    return $value^1$, $value^2$,...$value^n$

Examples (Writing the function being called):

```
def fun1 ():
    num1 = 10
    num2 = 20
    return num1, num2

def fun2 (num1, num2):
    num3 = num1 + num2
    return num3
```

Returning Values From Functions (2)

Format (Storing the return values):

variable[1], *variable*[2],...*variable*[n] = function name ()

Examples (Storing the return values):

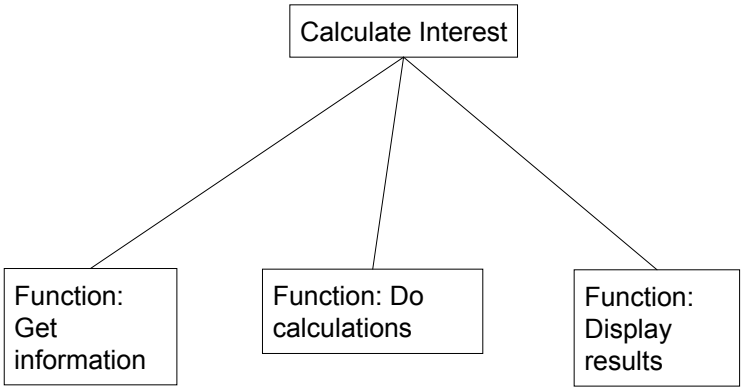
```
num1, num2 = fun1 ()

num3 = fun2 (num1, num2)
```


Parameter Passing And Return Values: An Example

Available online and is called "interest.py":

```
def calculate (principle, rate, time):
    interest = principle * rate * time
    total = principle + interest
    return interest, total

def start ():
    interest, total = calculate (100, 0.1, 5)
    print "Interest $ ", interest
    print "Total     $", total
```

Top Down Approach: Example Of Breaking A Programming Problem Down Into Parts (Functions)



Using Output Statements To Understand How A Program Works

- Example: what does the following program do?

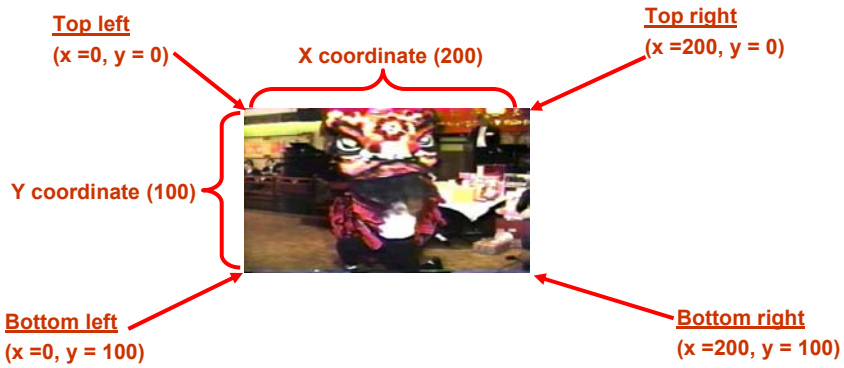
```
def start ():
    for i in range (1, 20, 1):
        j = i
        print "j=", j

        if (i % 5 == 0):
            print "i=", i
```

An output statement shows the successive values of 'i' inside the loop

An output statement shows the successive values of 'i' inside the loop and inside the if-branch

Information About Pictures In JES

- Images/pictures consist of pixels
- Each pixel has a set of coordinates (x, y) that determine it's location.
- Example image (200 pixels x 100 pixels)



Picture/Image-Related Functions In JES

- For more details look under the help menu under "Picture functions".
  - addLine (picture, startX, startY, endX, endY)
  - addRect (picture, startX, startY, width, height)
  - addRectFilled (picture, startX, startY, width, height, color)
  - addText (picture, xpos, ypos, text):

Explanation of the function values

- picture: the name of the picture that you want to edit
- startX: the starting pixel coordinate on the x axis
- startY: the starting pixel coordinate on the y axis
- endX: the ending pixel coordinate on the x axis
- endY: the ending pixel coordinate on the y axis
- width: the width of the rectangle in pixels
- weight: the height of the rectangle in pixels
- color: the color to fill the rectangle with (red, green, blue etc.)

Example Of Editing A Picture

- Available online and is called "picture1.py"

```
def picture1():
  lion = makePicture ("lion.jpg")
  addLine (lion,0,0,100,100)
  addRectFilled (lion,100,100,100,200,red)
  addText (lion,200,300,"Lion dance for the Lions Club")
  show (lion)
```

Important: Typically you want to show a picture only after you have finished all your edits!

Review: The 24 Bit Color Model

- Each pixel's color is specified with 24 bits:
  - 8 bits (256 combinations) for the red component
  - 8 bits (256 combinations) for the blue component
  - 8 bits (256 combinations) for the green component

- In JES the color value is an integer ranging from 0 – 255.
  - Smaller numbers result in darker pixels.
  - Larger numbers result in lighter pixels.

JES' Pixel-Related Functions

- Get functions: find out the color level of a particular pixel
  - getRed: returns the red level (0 – 255) of a pixel
  - getBlue: returns the blue level (0 – 255) of a pixel
  - getGreen: returns the green level (0 – 255) of a pixel
- Set functions: change the color of a particular pixel
  - setRed: change the red level of a pixel (to a value from 0 – 255)
  - setBlue: change the blue level of a pixel (to a value from 0 – 255)
  - setGreen: change the green level of a pixel (to a value from 0 – 255)

Example: Seeing The Color Values Of A Picture

- Available online and is called "picture2.py". It also requires that you download and save the picture "smallLion.jpg" into the folder that you run JES from.

```
def picture2 ():

  picture = makePicture ("smallLion.jpg")
  show (picture)
  allPixels = getPixels (picture)

  for pixel in allPixels:
    red = getRed (pixel)
    blue = getBlue (pixel)
    green = getGreen (pixel)
    print "RBG:", red, blue, green
```

Example: Changing The Color Values Of A Picture

- Available online and is called "picture3.py". It also requires that you download and save the picture "mediumLion.jpg" into the folder that you run JES from.

```
def picture3 ():
  picture = makePicture ("mediumLion.jpg")
  show (picture)
  allPixels = getPixels (picture)

  for pixel in allPixels:
    red = getRed (pixel)
    blue = getBlue (pixel)
    green = getGreen (pixel)
    if ((red + 50) <= 255):
      setRed (pixel, (red+50))
    if ((blue + 50) <= 255):
      setBlue (pixel, (blue+50))
    if ((green + 50) <= 255):
      setGreen (pixel, (green+50))
  repaint (picture)
```

Show the original picture loaded from file.

Show the original picture after it has been manipulated.

You Should Now Know

- What is the purpose of a loop
- How to write a for-loop in JES
- How functions are a way of implementing a top down approach to program design
- How to write and trace through a program that employs functions
- How to pass information to and from functions via parameters and return values
- A technique for tracing programs: using output statements
- Functions for working with images/pictures in JES