CSC 221: Introduction to Programming

Fall 2018

Python control statements

- counter-driven repetition: for
- conditional repetition: while
- simulations & modeling
  - counters, sums
- functions vs. methods
  - Turtle, Pixel, Image
- image manipulation

Repetition

an if statement provides for conditional execution

- can make a choice between alternatives, choose which (if any to execute)

if we want to repeatedly execute a block of code, need a loop

- loops can be counter-driven
  e.g., roll a die 10 times

- loops can be condition-driven
  e.g., roll dice until doubles

the simplest type of Python loop is a *counter-driven* for loop

```
for i in range(NUM_REPS):
   STATEMENTS_TO_BE_REPEATED
```

For loop examples

```
>>> for i in range(5):
        print(flipCoin())


heads
tails
tails
tails
heads
```

```
>>> for i in range(6):
        print(i, i**2, i**3)


0 0 0
1 1 1
2 4 8
3 9 27
4 16 64
5 25 125
```

```
def roll_die(num_sides):
    """Simulates rolling a die with num_sides."""
    return random.randint(1, num_sides)
```

```
>>> for _ in range(10):
        print(roll_die(8))


2
6
7
8
5
6
7
8
6
8
>>> for _ in range(10):
        roll = roll_die(6) + roll_die(6)
        print(roll)


8
2
3
6
9
8
2
11
10
11
```

3

For loops & ranges

technically, a for loop ranges over a *sequence* of values

- the simplest sequences are lists of values in brackets, separated by commas

```
[0, 1, 2, 3, 4]

["yes", "no", "maybe"]

[True, False]
```

recall that the range function returns a sequence

e.g., range(5) → [0, 1, 2, 3, 4] range(3, 6) → [3, 4, 5]

```
for i in range(5):
    print(i)
```

is same as

```
for i in [0, 1, 2, 3, 4]:
    print(i)
```

4

Exercise: sum the dice rolls

suppose we wanted to define a function to sum up dice rolls

- need to initialize a variable to keep track of the sum (starting at 0)
- inside the loop, add each roll to the sum variable
- when done with the loop, display the sum

```
def sum_rolls(num_rolls):
    ???
```

similarly, suppose we wanted to average the dice rolls

- calculate the sum, as before
- return sum divided by the number of rolls

```
def avg_rolls(num_rolls):
    ???
```

5

Loops & counters

for loops can be combined with if statements

- common pattern: perform multiple repetitions and count the number of times some event occurs
- e.g., flip a coin and count the number of heads
- e.g., roll dice and count the number of doubles
- e.g., traverse an employee database and find all employees making > $100K

```
def count_heads(num_flips):
    """Counts the number of heads obtained
       from num_flips coin flips."""
    num_heads = 0
    for i in range(num_flips):
        flip = flip_coin()
        if flip == "heads":
            num_heads += 1
    return num_heads
```

```
def count_doubles(num_rolls):
    """Counts the number of doubles obtained
       from num_rolls dice rolls."""
    num_doubles = 0
    for i in range(num_rolls):
        roll1 = roll_die(6)
        roll2 = roll_die(6)
        if roll1 == roll2:
            num_doubles += 1
    return num_doubles
```

6

Shorthand assignments

a variable that is used to keep track of how many times some event occurs is known as a *counter*

- a counter must be initialized to 0, then incremented each time the event occurs

```
def count_heads(num_flips):
    """Counts the number of heads obtained
       from num_flips coin flips."""
    num_heads = 0
    for i in range(num_flips):
        flip = flip_coin()
        if flip == "heads":
            num_heads += 1
    return num_heads
```

shorthand notation

```
number += 1  ↔  number = number + 1
number -= 1  ↔  number = number - 1
```

other shorthand assignments can be used for updating variables

```
number += 5  ↔  number = number + 5
number *= 2  ↔  number = number * 2
```

7

While loops

the other type of repetition in Python is the *condition-driven* while loop

- similar to an if statement, it is controlled by a Boolean test
- unlike an if, a while loop *repeatedly* executes its block of code as long as the test is true

```
while TEST_CONDITION:
  STATEMENTS_TO EXECUTE_AS_LONG_AS_TEST_IS_TRUE
```

```
>>> flip = ""
>>> while flip != "heads":
        flip = flip_coin()
        print(flip)
```

```
>>> roll1 = -1
>>> roll2 = -2
>>> while roll1 != roll2:
        roll1 = roll_die(6)
        roll2 = roll_die(6)
        print(roll1, roll2)
```

8

Example: hailstone sequence

interesting problem from mathematics

- start a sequence with some positive integer N
- if that number is even, the next number in the sequence is N/2;
  if that number is odd, the next number in the sequence is 3N+1

5 → 16 → 8 → 4 → 2 → 1 → 4 → 2 → 1 → …

15 → 46 → 23 → 70 → 35 → 106 → 53 → 160 → 80 → 40 → 20 → 10 ↓

… ← 1 ← 2 ← 4 ← 8 ← 16 ← 5

it has been conjectured that no matter what number you start with, you will end up stuck in the 4-2-1 loop

- has been shown for all values <= $20 \times 2^{58} \approx 5.764 \times 10^{18}$
- but has not been proven to hold in general

9

Generating a hailstone sequence

need to be able to distinguish between even and odd numbers

- recall the remainder operator, %
- (x % y) evaluates to the remainder after dividing x by y

- thus, (x % 2) evaluates to 0 if x is even, 1 if x is odd

```
def hailstone(num):
    """Displays the hailstone sequence starting at num."""
    print(num)
    while num != 1:
        if num%2 == 0:
            num = num // 2
        else:
            num = 3*num + 1
        print(num)
```

EXERCISE: modify so that it also prints the length of the sequence

10

Beware of "black holes"

since while loops repeatedly execute as long as the loop test is true, infinite loops are possible (a.k.a. *black hole* loops)

```
def flip_until_heads():
    """Simulates repeatedly flipping
       a coin until get heads."""
    flip = flip_coin()
    num_flips = 1
    print(num_flips, ":", flip)
    while flip != "heads":
        num_flips += 1
        print(num_flips, ":", flip)
```

PROBLEM?

- a necessary condition for loop termination is that some value relevant to the loop test must change inside the loop

  in the above example, `flip` doesn't change inside the loop

  →if the test succeeds once, it succeeds forever!

- is it a sufficient condition? that is, does changing a variable from the loop test guarantee termination?

  NO – "With great power comes great responsibility."

  fix to above function?

11

Example: Pig

Pig is a 2-player dice game in which the players take turns rolling a die.

On a given turn, a player rolls until either

1. he/she rolls a 1, in which case his/her turn is over and no points are awarded, or
2. he/she chooses to hold, in which case the sum of the rolls from that player's turn are added to his/her score.

The winner of the game is the first player to reach 100 points.

for example:

SCORE = 0 to start
TURN 1: rolls 5, 2, 4, 6, holds → SCORE += 17 → 17
TURN 2: rolls 4, 1, done → SCORE += 0 → 17
TURN 3: rolls 6, 2, 3, hold → SCORE += 11 → 28
…

12

Pig simulation

we want to simulate Pig to determine the best strategy

- i.e., determine the optimal cutoff such that you should keep rolling until the score for a round reaches the cutoff, then hold
- i.e., what is the optimal cutoff that minimizes the expected number of turns

```
def pig_turn(cutoff):
    """Simulates a turn of the game Pig, with the player repeatedly
       rolling until they get a 1 or their score reaches the cutoff."""
    score = 0
    roll = 0
    while roll != 1 and score < cutoff:
        roll = roll_die(6)
        if roll == 1:
            score = 0
        else:
            score += roll
        print(roll, "-->", score)
```

why is roll set to 0 before the loop?
why not set it to roll_die(6)?

EXERCISE: modify the `pig_turn` function so that it returns the score for the round (as opposed to printing rolls/scores)

13

Pig simulation (cont.)

EXERCISE: define a `pig_game` function that simulates a Pig game

- has 1 input, the cutoff value for each turn
- it repeatedly calls the `pig_turn` function, totaling up the score for each turn (and displaying the turn # and updated score)
- it stops when the score total reaches 100

```
>>> pig_game(15)
Turn 1 : 0
Turn 2 : 0
Turn 3 : 0
Turn 4 : 0
Turn 5 : 0
Turn 6 : 17
Turn 7 : 33
Turn 8 : 33
Turn 9 : 52
Turn 10 : 52
Turn 11 : 52
Turn 12 : 69
Turn 13 : 69
Turn 14 : 69
Turn 15 : 88
Turn 16 : 101
```

```
>>> pig_game(20)
Turn 1 : 0
Turn 2 : 22
Turn 3 : 22
Turn 4 : 45
Turn 5 : 45
Turn 6 : 66
Turn 7 : 66
Turn 8 : 66
Turn 9 : 66
Turn 10 : 86
Turn 11 : 86
Turn 12 : 86
Turn 13 : 86
Turn 14 : 101
```

```
>>> pig_game(20)
Turn 1 : 0
Turn 2 : 0
Turn 3 : 0
Turn 4 : 0
Turn 5 : 23
Turn 6 : 23
Turn 7 : 23
Turn 8 : 23
Turn 9 : 43
Turn 10 : 43
Turn 11 : 43
Turn 12 : 43
Turn 13 : 43
Turn 14 : 43
Turn 15 : 43
Turn 16 : 43
Turn 17 : 43
Turn 18 : 43
Turn 19 : 43
Turn 20 : 43
Turn 21 : 66
Turn 22 : 66
Turn 23 : 86
Turn 24 : 100
```

14

Pig simulation (cont.)

what can we conclude from running several experiments?

- Simulation 1: a cutoff of 15 yields a game of 16 turns
- Simulation 2: a cutoff of 20 yields a game of 14 turns
- can we conclude that a cutoff of 20 is *better* than a cutoff of 15?

note: because of the randomness of the die, there can be wide variability in the simulations

- note: a single roll of a die is unpredictable
- however: given a *large* number of die rolls, the distribution of the rolls can be predicted (since each die face is equally likely, each should appear ~ 1/6 of time)
- *Law of Large Numbers* states that as the number of repetitions increases to ∞, the percentages should get closer and closer to the expected values

15

Pig simulation (cont.)

in order to draw reasonable conclusions, will need to perform many experiments and average the results

EXERCISE: modify the `pig_game` function so that it returns the number of turns (as opposed to printing turns/scores)

EXERCISE: define a `pig_stats` function that simulates numerous games

- has 2 inputs, the number of games and the cutoff value for each turn
- it repeatedly calls the `pig_game` function the specified number of times, totaling up the number of turns for each game
- it returns the average number of turns over all the games

QUESTION: what is the optimal cutoff that minimizes the number of turns

- how many games do you need to simulate in order to be confident in your answer?

16

Control summary

if statements provide for conditional execution

- use when you need to make choices in the code
- control is based on a Boolean (True/False) test
  - 1-way: if (with no else)
  - 2-way: if-else
  - multi-way: cascading if-else, if-elif-elif-…-elif-else

for loops provide for counter-driven repetition

- use when you need to repeat a task a set number of times
- utilizes the range function (or, in rare cases, a specific sequence of values)

while loops provide for conditional repetition

- use when you need to repeat a task but you don't know how many times
- control is based on a Boolean (True/Falsе) test
- as long as test continues to be True, the indented code will be executed
- beware of infinite (black hole) loops

17

Functions vs. methods

typically, you call a function by specifying the name and any inputs in parentheses

```
int("12")     float("34.5")     str(300)     abs(-4)
```

- if the function is defined in a module, use . to specify the module

```
math.sqrt(36)     random.randint(1, 6)
```

however, turtles were different

```
import turtle

alex = turtle.Turtle()

alex.forward(20)
```

Turtle is a new data type known as a *class* (more details later)

- values of a class are known as *objects*
- each object automatically has functions associated with it
- these special functions, called *methods*, are called by specifying `object.func(. . .)`

18

Image representation

the online book provides a pair of classes that are useful for manipulating images: Pixel & Image

first, some background on image representation

- images are stored using a variety of formats and compression techniques
- the simplest representation is a *bitmap*
- bitmaps partition an image into a grid of picture elements, called *pixels*, and then convert each pixel into a bit pattern
- resolutions refers to the number of pixels per square inch (more pixels → higher resolution → more storage)

1. To generate a bitmap, the image is first partitioned into a grid of pixels, here 8 × 8.

2. For a black-and-white bitmap, a black pixel is represented with a 0 bit, a white pixel with a 1 bit.

19

Color formats

when creating a bitmap of a color image, more than one bit is required to represent each pixel

- the most common system is to translate each pixel into a 24 bit code, known as its *RGB value*: 8 bits to represent the intensity of each red/green/blue component

Common HTML Colors

color	(R, G, B)	color	(R, G, B)	color	(R, G, B)
red	(255, 0, 0)	green	(0, 128, 0)	blue	(0, 0, 255)
darkred	(139, 0, 0)	darkgreen	(0, 100, 0)	darkblue	(0, 0, 139)
maroon	(128, 0, 0)	forestgreen	(34, 139, 34)	royalblue	(65, 105, 225)
crimson	(220, 20, 60)	olive	(128, 128, 0)	lightblue	(173, 216, 230)
pink	(255, 192, 203)	lightgreen	(144, 238, 144)	purple	(128, 0, 128)
violet	(238, 130, 238)	brown	(165, 42, 42)	gray	(128, 128, 128)
orange	(255, 165, 0)	white	(255, 255, 255)	black	(0, 0, 0)

common image formats implement various compression techniques to reduce storage size

- GIF (Graphics Interchange Format)
  - a *lossless* format, meaning no information is lost in the compression
  - commonly used for precise pictures, such as line drawings
- PNG(Portable Network Graphics)
  - more modern alternative to GIF - more colors, 10-50% more compact
- JPEG (Joint Photographic Experts Group)
  - a *lossy* format, so the compression is not fully reversible (but more efficient)
  - commonly used for photographs

20

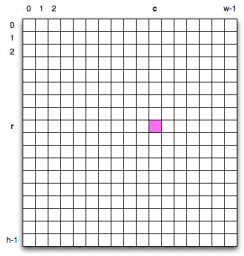
Pixel and Image classes

Method Name	Example	Explanation
Pixel(r,g,b)	Pixel(20,100,50)	Create a new pixel with 20 red, 100 green, and 50 blue.
getRed()	r = p.getRed()	Return the red component intensity.
getGreen()	r = p.getGreen()	Return the green component intensity.
getBlue()	r = p.getBlue()	Return the blue component intensity.
setRed()	p.setRed(100)	Set the red component intensity to 100.
setGreen()	p.setGreen(45)	Set the green component intensity to 45.
setBlue()	p.setBlue(156)	Set the blue component intensity to 156.

Method Name	Example	Explanation
Image(filename)	img = image.Image("cy.png")	Create an Image object from the file cy.png.
EmptyImage()	img = image.EmptyImage(100,200)	Create an Image object that has all "White" pixels
getWidth()	w = img.getWidth()	Return the width of the image in pixels.
getHeight()	h = img.getHeight()	Return the height of the image in pixels.
getPixel(col,row)	p = img.getPixel(35,86)	Return the pixel at column 35, row 86.
setPixel(col,row,p)	img.setPixel(100,50,mp)	Set the pixel at column 100, row 50 to be mp.

the author provides classes for representing pixels and images

see chapter 8.10



21

Simple image access

the interactive book has several images you can choose from

- once you create an image, you can…
  - get its dimensions
  - get the RGB for a specific pixel

Run

```
import image
img = image.Image("cy.png")

print(img.getWidth())
print(img.getHeight())

p = img.getPixel(45, 55)
print(p.getRed(), p.getGreen(), p.getBlue())

p = img.getPixel(100, 100)
print(p.getRed(), p.getGreen(), p.getBlue())

```

```
228
284
255 255 255
220 7 97
```

22

Nested loops

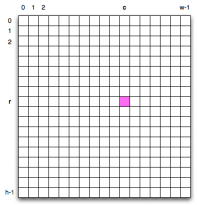
to traverse a 2-dimensional structure (such as an image), need nested loops

```
>>> for row in range(3):
        for col in range(4):
            print(col, row)


0 0
1 0
2 0
3 0
0 1
1 1
2 1
3 1
0 2
1 2
2 2
3 2
```

```
>>> for col in range(4):
        for row in range(3):
            print(col, row)


0 0
0 1
0 2
1 0
1 1
1 2
2 0
2 1
2 2
3 0
3 1
3 2
```

23

Manipulating images



can similarly go through the rows and columns of an image

- # of rows is `img.getHeight()`
- # of cols is `img.getWidth()`

```
for row in range(img.getHeight()):
    for col in range(img.getWidth()):
        p = img.getPixel(col, row)

        # determine RGB value of newpixel

        img.setPixel(col, row, newpixel)
```

24

Image negative

to get a photo negative effect, invert each RGB value

- e.g., (0, 0, 0) → (255, 255, 255) (100, 0, 100) → (155, 255, 155)

Run

```
import image

img = image.Image("cy.png")
win = image.ImageWin(img.getWidth(), img.getHeight())
img.draw(win)
img.setDelay(1,30)   # setDelay(0) turns off animation

for row in range(img.getHeight()):
    for col in range(img.getWidth()):
        p = img.getPixel(col, row)

        newred = 255 - p.getRed()
        newgreen = 255 - p.getGreen()
        newblue = 255 - p.getBlue()

        newpixel = image.Pixel(newred, newgreen, newblue)

        img.setPixel(col, row, newpixel)

img.draw(win)
win.exitonclick()
```

25

Grayscale

to make an image grayscale, assign each RGB value to the average

- e.g., (40, 120, 200) → (120, 120, 120)

Run

```
import image

img = image.Image("cy.png")
win = image.ImageWin(img.getWidth(), img.getHeight())
img.draw(win)
img.setDelay(1,30)   # setDelay(0) turns off animation

for row in range(img.getHeight()):
    for col in range(img.getWidth()):
        p = img.getPixel(col, row)

        avg = round((p.getRed()+p.getGreen()+p.getBlue())/3)

        newpixel = image.Pixel(avg, avg, avg)

        img.setPixel(col, row, newpixel)

img.draw(win)
win.exitonclick()
```

26

Black & white

to make an image black & white, for each pixel:

- calculate the average RGB value – same as with grayscale
- if avg < 128 → (0,0,0)
- if avg >= 128 → (255,255,255)

EXERCISE: modify the code (chapter 10.8, ActiveCode: 4) to convert the image to black & white

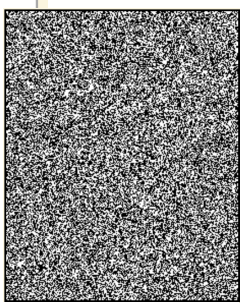
27

Mystery code

what will the following code do?

- note: it does not even look at the original value of each pixel

```
1  import image
2  import random
3
4  img = image.Image("cy.png")
5  win = image.ImageWin(img.getWidth(), img.getHeight())
6  img.draw(win)
7  img.setDelay(1,30)   # setDelay(0) turns off animation
8
9  for row in range(img.getHeight()):
10     for col in range(img.getWidth()):
11         pick = random.choice([0,255])
12
13         newpixel = image.Pixel(pick, pick, pick)
14
15         img.setPixel(col, row, newpixel)
16
17 img.draw(win)
18 win.exitonclick()
19
```



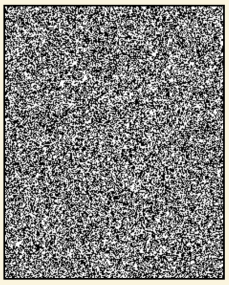
replaces each pixel with a random black or white

28

Exercises

Run

```
import image
import random

img = image.Image("cy.png")
win = image.ImageWin(img.getWidth(), img.getHeight())
img.draw(win)
img.setDelay(1,30)   # setDelay(0) turns off animation

for row in range(img.getHeight()):
    for col in range(img.getWidth()):
        pick = random.choice([0,255])

        newpixel = image.Pixel(pick, pick, pick)

        img.setPixel(col, row, newpixel)

img.draw(win)
win.exitonclick()

```



1. modify this code so that it generates a *random* grayscale image
   each pixel is of the form (V,V,V) where $0 \leq V \leq 255$

2. modify this code so that it generates a *random* color image
   each pixel is of the form (R,G,B) where each component $0 \leq R/G/B \leq 255$

29

Horizontal reflection

to reflect horizontally, replace every pixel in the bottom half of the image with the corresponding pixel from the top half

Run

```
import image
import random

img = image.Image("cy.png")
win = image.ImageWin(img.getWidth(), img.getHeight())
img.draw(win)
img.setDelay(1,30)   # setDelay(0) turns off animation

for row in range(img.getHeight()//2):
    for col in range(img.getWidth()):
        p = img.getPixel(col, row)

        img.setPixel(col, img.getHeight()-row-1, p)

img.draw(win)
win.exitonclick()

```

30

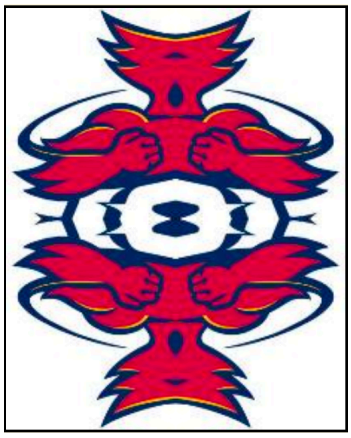
Vertical reflection

EXERCISE: modify the code so that it performs a vertical reflection

31

Double reflection

EXERCISE: modify the code so that it performs a double reflection



32