Finding Pi:
Applications of Loops, Random Numbers, Booleans

CS 8: Introduction to Computer Science, Winter 2018
Lecture #6

Ziad Matni
Dept. of Computer Science, UCSB

Administrative

- New Homework (#3) is online – due next Monday
- New Lab (#2) this week

- The grader, Vivek Pradhan, will hold office hours on **Thursdays from 3pm to 4pm** in the **CSIL lab**

- Midterm is on **Wednesday, Feb 14th**

Lecture Outline

- More Loops with Mathematical Applications
  - Looking for Pi in all the Right Places…
- Characters and Strings in Python

Class Exercise

Get together with 2 or 3 other people around you and answer this question.
You can use your notes from other lectures:

Write Python code that asks a user for a number between 0 and 200 (inclusive). Call that variable **SideParam**. Then it asks the user for *another* number between 3 and 360 (inclusive). Call that variable **Sides**.

a) The code should first **check** that the 2 input numbers do indeed meet this criteria. Bonus points if you can do this with **ONE if-statement**! If any of the criteria is not met, you should print a message to say it was out of range and stop.

b) Then, have your code use the Turtle graphics module to draw a ***polygon*** with number of sides equal to **Sides** and having side lengths to half of **SideParam**, plus 50.

An Ancient Problem: Finding 

- Ratio of a circle's circumference to its diameter

```
π  = circumference / diameter   # for any circle
```

- Irrational number: an infinite series of non-repeating digits
  - So it can never be represented exactly, only *approximated*
- Chapter 2 explores various ways to approximate pi
  - But just to teach problem-solving. For calculating, use `math.pi` module

```
import math # necessary to use math module
area = math.pi * radius * radius
```

- By the way, the math module has lots of other cool stuff
  - Square root, trig functions, e, … for more info on IDLE, try `>>> help(math)`

The `math` Library

Must be **`import`**-ed

Contains lots of often-used mathematical functions, like:

- `math.fabs(x)` `# Returns the absolute value of x`
- `math.exp(x)` `# Returns e**x`
- `math.pow(x,y)` `# Returns x**y`
- `math.sqrt(x)` `# Returns the square-root of x`
- `math.log(x, b)` `# Returns the log of x, base b`
- `math.sin(x)` ***or*** `.cos(x)` ***or*** `.tan(x)` `# Trig functions`
- `math.pi` `# Returns pi (3.141…)`
- `math.e` `# Returns e (2.718…)`

- See https://docs.python.org/2/library/math.html for full details

CLASS DEMO

Accumulator Pattern

- We can calculate PI using summing infinite series
  - General idea applies to counting, summing, …
- Idea: **set initial value**, then **loop to update a running sum**
  - e.g., add numbers 1 through 5:

```
sum = 0        # initialize sum (accumulator variable)
for number in range(1, 6):
    sum = sum + number # updates sum
```

- See textbook for 2 different ways to find pi:
  - Leibniz Formula – summation of terms (p.58) --- ACCUMULATED SUM
  - Wallis Formula – product of terms (p. 60) --- ACCUMULATED PRODUCT

Liebniz Formula

$$1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots = \frac{\pi}{4}$$

- So, the formula suggests that:

$\pi = 4 \cdot \{ \Sigma (-1)^n \cdot [1 / (2n + 1)] \}$ *as n goes from 0 → ∞*

- When n = 0, $\pi_{est} = 4 \cdot (1) = 4$
- When n = 1, $\pi_{est} = 4 \cdot (1 - 1/3) = 8/3 = 2.66667$
- When n = 2, $\pi_{est} = 4 \cdot (1 - 1/3 + 1/5) = 3.46667$

…

- When n = 100, $\pi_{est} = 4 \cdot (1 - 1/3 + 1/5 + \ldots + 1/201) = 3.13159$

CLASS DEMO: HOW TO CODE THIS!

Accumulated Product

- Example: How would you create a function that takes a positive integer N and returns the product of all numbers less than or equal to N?

- In other words: Product(N) = 1 x 2 x 3 x … N

- Example: Product(3) = 6,
  Product(4) = 24, etc…

CLASS DEMO: HOW TO CODE THIS!

Random Values

Must be **`import`**-ed

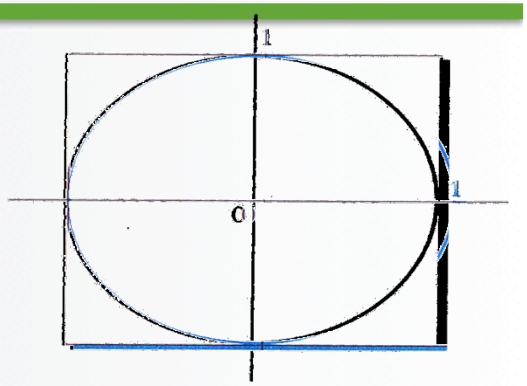
- "Pseudo-random" values can be generated using special functions in most programming languages

- In Python use functions of the `random` module
  - Simplest is `random.random()` – returns a floating point value between 0.0 and 1.0
  - Also `randrange(n)`, `randint(low, high)`, `shuffle(list)` and many others
  - Try `help(random)` to learn more … and *play around* with it

- For example, Listing 2.5 uses `random()` for x, y dart locations

CLASS DEMO: HOW TO USE random

Monte Carlo Simulation



- A popular statistical method using randomness to solve problems.
  - Used in many simulation – traffic flows, length of bank queues, etc…
- In the case of estimating pi – imagine throwing darts at a unit circle (i.e. `r = 1`) inscribed inside a square (i.e. whose side = 2r = 2)
  - Circle area = $\pi r^2 = \pi$
  - Square area = `2 * 2 = 4`
  - So if **`n`** darts hit the square, how many darts (`k`) should land inside the circle by chance alone?
  - As it turns out, that's proportional to the area of the circle divided by the area of the square.
  - Answer: `k = n * π/4`. In other words, we can approximate π_{est} `= 4 * k/n`

See Listing 2.5 in textbook

CLASS DEMO: HOW TO USE random

montePi(numDarts)

```
def montePi(numDarts):
                # numDarts is the number of darts that we throw at the square
    k = 0       # k is the nuber of darts that hit the circle inside the square

    for i in range(numDarts):
        x = random.random()             # x and y are random coordinates
        y = random.random()             # representing the dart throw location
        d = math.sqrt(x**2 + y**2)      # d = distance between (x,y) and origin (0,0)
        if d <= 1:                      # if d <= 1, it means that the
            k = k + 1                   # hit is within the circle, so count those

    pi = 4 * (k /numDarts)
    return pi
```

QUESTION: How close do we get to actual π using this method?
(see demo from class…)

Boolean Expressions

- Expressions that evaluate to `True` or `False`
- Relational operators: `<` `<=` `>` `>=` `==` `!=`

  Example: `9 > 7 is True`, while `(4.5 - 3) >= (3 - 1.3) is False`

- Watch out when using `==` or `!=` with ***floating point numbers***

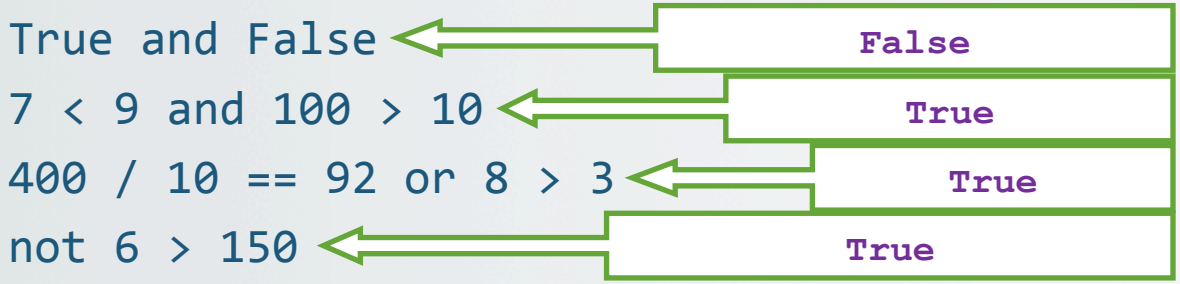
  Example: `100/3 == 33.3333` ← `False (why?)`

  – Instead it's better to compare absolute difference to a small value

  `abs(100/3 - 33.3333) < 0.0001` ← `True (why?)`

Compound Boolean Expressions

- Logical operators: `and, or, not`
- Their operands are Boolean values:



- Special Python feature: `low <= value <= high`
- The special role that 0 and 1 play
  - See other behavior notes in Table 2.2 (p. 66)

YOUR TO-DOs

- ❑ Finish reading **Chapters 2** and **3** for next class
- ❑ Finish **Homework3** (due **Monday 2/12**)
- ❑ Finish **Lab2** (due **Wednesday 2/7**)

- ❑ Run through an open meadow

</LECTURE>