Objective

- More Functions
- Animation
- `random` Module

1

Review

Reminder: Review slides and notes from last time

- What is the *accumulator design pattern*?
- What are some variations in how we use the `print` function?
- What are benefits of functions?
- What is a *module*?
  - What are some available modules? What functionality do they have?
  - How can we find out what functionality is in a module?
- How can we access the functionality defined in the modules (two ways)?
  - How does that choice affect how we use the functionality in our code?

2

Review: Accumulator Design Pattern

1. Initialize accumulator variable
2. Loop until done
   - Update the value of the accumulator
3. Display result

3

Review: Using `print`

- `print(*objects, sep=' ', end='\n', file=sys.stdout)`

  Semantics: default values for `sep` is `' '` and `end` is `'\n'`

- Examples:

```
print("Hi", "there", "class", sep='; ')
print("Put on same", end='')
print("line")
```

Output:

```
Hi; there; class
Put on sameline
```


4

Review: Comparing Import Statements

`import <module>`

- Requires prepending constants/functions with module
  - Ex: `math.sqrt(num)`
- Benefits:
  - Helps you to know which module the constant/function is coming from
  - No problem with name clashes if two modules define the same function
    - math.aFunction()
    - os.aFunction()

`from <module> import <defn_name>`

- Don't need to prepend constants/functions with module
  - Ex: `sqrt(num)`
- Benefit: Easier to write/read

5

Review: Benefits of Functions

- Allows us to reuse, reduce code
  - Don't need to rewrite code that has already been defined
- Breaks problems into more manageable pieces
- Easier to read, write
- From Python modules: know they work and are efficient

6

Review: for loop analysis

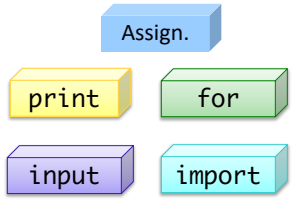
```
for i in range(5):
      # like assigning i values(0,1,2,3,4)
      # consecutively, each time through loop

      # rest of loop body …
```

- When we have range(5),
  - i is set to the values (0, 1, 2, 3, 4)
  - Which means that loop executes 5 times
- Optional: start and step parameters

7

Programming Building Blocks

- Adding to your tool set
- We can combine them to create more complex programs
  - Solutions to problems



8

Moving a Circle According to the User

- Draw a circle in the upper left-hand corner of the screen
- Tell the user to click somewhere
- Move the circle to where the user clicked

Hmm…. Some of these steps seem very different from what we've been doing.
Can we even do them?
How can we figure out if we can?

9

ANIMATION

10

Animation

- Use combinations of the method **`move`** and the function **`sleep`**
  - Need to **sleep** so that humans can see the graphics moving
  - Otherwise, computer processes the **move**s too fast!
- **`sleep`** is part of the **`time`** module
  - takes a float representing *seconds* and pauses for that amount of time

`animate.py`

Animate Circle Shift!

- Animate moving a circle to the position clicked by the user
  - Previously, moved in one fell swoop

```
dx = newX - circle.getCenter().getX()
dy = newY - circle.getCenter().getY()

circle.move(dx, dy)
```

  - To animate
    - Break the movement into chunks
    - Repeatedly, move one chunk, sleep
- Bonus: do the user clicks, animation 3 times

 `circleShiftAnim.py`

Examples of Animation

- From Previous Classes

`random` module

- Python provides the `random` module to generate pseudo-random numbers
- What is "pseudo-random"?
  - Generates a list of random numbers and grabs the next one off the list
  - A ***seed*** is used to initialize the random number generator, which decides which list to use
    - By default, the current time is used as the seed

List of Lists of Random Numbers

Seed	List of Random Numbers				
1	0.1343642441	0.8474337369	0.763774619	0.2550690257	…
2	0.9560342719	0.9478274871	0.0565513677	0.0848719952	…
3	0.2379646271	0.5442292253	0.3699551665	0.6039200386	…
4	0.2360480897	0.1031660342	0.3960582426	0.1549722708	…
…		…			…

`random_test.py`

15

Why use "pseudo-random" numbers?

- No cost-effective source of real randomness
- Code usually doesn't *need* to be truly random
- Can replicate the code that depends on randomness by using the seed, when appropriate

16

Some `random` Functions

- `random()`
  - Returns the next random floating point number in the range [0.0, 1.0)
- `randint(a, b)`
  - Return a random integer N such that a ≤ N ≤ b

```
import random

#random.seed(1)     # module.function()

for x in range(10):
    print(random.random())
```

random_test.py

VA Lottery: Pick 4

- To play: pick 4 numbers between 0 and 9
- To win: select the numbers that are selected by the magic ping-pong ball machine
- Your job: Simulate the magic ping-pong ball machines
  - Display the number on *one* line

pick4.py

Looking Ahead

- Pre Lab 3 due before lab
- Lab 3 due Friday
- Broader Issue on Chat GPT for Friday