

# CIS 122

Final Review (part 2)

# Types

- Integers
- Floats
- Strings
- Booleans
- Lists
  - Nested Lists
- Dictionaries

# Programming Concepts

- Variables
- Functions
- Conditionals
- Recursion
- Iteration
  - Nested Loops
- Classes

# Variables

- Store values
- Define using assignment operator ( = )
  - `color = "blue"`
  - `x = 5`
- Reassign previously assigned variables
  - `color = "red"`
  - `x = x + 1`
  - `x += 1`

# Variables

- Reassigning variable does not change object

```
num1 = 5
```

```
num2 = num1
```

```
num1 = 6
```

```
print num2
```

- Modifying an object does

```
list1 = [1,2,3]
```

```
list2 = list1
```

```
list1.append(4)
```

```
print list2
```

# Functions

- Blocks of code
  - Take input (zero or more arguments)
  - Return output

```
def addOne(myNum):  
    nextNum = myNum + 1  
    return nextNum
```

- What happens when we call a function?

```
>>> x = addOne(5)      myNum → 5  
                          nextNum → 6 x → 6
```

# Conditionals

- Conditionally execute blocks of code
  - `if`
  - `elif`
  - `else`

```
if x > 90:  
    return "A"  
elif x > 80:  
    return "B"  
elif x > 70:  
    return "C"  
else:  
    return "D"
```

# Recursion

- A function which solves a problem by calling itself
  - Solving a smaller version of the problem
- Base Case
  - Some trivial case
  - Solve for 0
  - Solve for empty list
- Recursive Step
  - Solve problem by calling function again
  - Reduce problem towards base case



# Recursion

- Define a function `count(L, element)`
  - Return number of times element occurs in L
- Base Case
  - element never occurs in the empty list
- Recursive Step
  - Check the first element of the list
  - Check the rest of the list
  - Return the sum

# Iteration

- Repeating the same block of code over and over
- Two kinds of loops
- for loop
  - Keep looping for each element in a sequence
  - Good for well specified loops
- while loop
  - Keep looping while some condition is true
  - Good for indeterminate loops

# Iteration

- for loops
- Good for iterating directly over sequences
  - `for char in string:`
  - `for element in list`
- Good for repeating a task a certain number of times
  - `for i in range(10):`
- Good for iterating over indices
  - `for i in range(len(string))`  
`print string[i]`

# Iteration

- while loops
- Good for arbitrarily long loops
  - `while True:`
  - `while game.allOff() == False:`
- If you can't phrase it as a for loop, use a while loop

# Iteration

- Define a function  $\text{count}(L, \text{element})$ 
  - Return number of times element occurs in L
- Set up a tally
- Loop through L examining each element
  - Increment tally if necessary
- After loop, return the tally
- What sort of loop should we use?

# Nested Loops

- To examine all the elements in a nested list
  - You need a nested loop

```
nestedList = [ [10, 20, 30, 40],  
               [11, 21, 31, 41],  
               [12, 22, 32, 42],  
               [13, 23, 33, 43] ]
```

```
for row in nestedList:  
    print row
```

# Nested Loops

- To examine all the elements in a nested list
  - You need a nested loop

```
nestedList = [ [10, 20, 30, 40],  
               [11, 21, 31, 41],  
               [12, 22, 32, 42],  
               [13, 23, 33, 43] ]
```

```
for row in nestedList:  
    for element in row:  
        print element
```

# Nested Loops

- To examine all the elements in a nested list
  - You need a nested loop

```
nestedList = [ [10, 20, 30, 40],  
               [11, 21, 31, 41],  
               [12, 22, 32, 42],  
               [13, 23, 33, 43] ]
```

```
for row in range(len(nestedList)):  
    for col in range(len(nestedList[0])):  
        print nestedList[row][col]
```



# Classes

- Custom Types
  - Collection of attributes and methods
- Attributes - nouns
  - grid
  - numRows
- Methods - verbs
  - toggle
  - press

# Classes

- Class methods
  - Special first argument
  - Refers to object calling method

```
def toggle(self, row, col):  
    <code goes here>
```

```
>>> game = LightsOut()  
>>> game.toggle(3, 5)
```

```
self → game  
row → 3  
col → 5
```

# Classes

- Important Methods
- `__init__(self)`
  - Constructor
  - Instantiates a new object (but does not return it)
  - Called with `ClassName()`
- `__repr__(self)`
  - Print method
  - Returns string representation of object
  - Called whenever object is printed

# Classes

- Important Methods
- `__cmp__(self, other)`
  - Comparison method
  - Returns a number
    - Positive if `self > other`
    - Negative if `self < other`
    - 0 if `self == other`
  - Called whenever two objects are compared