CS 21 Final Review

About the Final

- Saturday, 7-10pm in Science Center 101
- Closed book, closed notes
- Not on the final: graphics, file I/O, vim, unix

Expect Questions That Ask You To:

- Evaluate Python expressions and identify their types
- Trace a program, drawing stack diagram and showing output
- Write a complete Python program with multiple functions
- Write recursive functions, draw the stack for a recursive function

Expect Questions That Ask You To:

- Trace the execution of searching or sorting algorithms, identify their big O run times

- Write methods for a class, look at a class definition and understand how to use it.

- Write/understand methods for LinkedList class, use LinkedList class, draw structure of a linked list, identify big O run times for operations on Python lists and linked lists.

- Find and fix bugs in a function

Types

- int
- float
- bool
- string (also a sequence)
- list (also a sequence)
- None
- objects belonging to various classes

Arithmetic Operators

- Addition / Concatenation: +
- Multiplication / Replication: *
- Subtraction: -
- Division: /
  - Integer division
- Remainder: %

★ Promotion: an int gets **promoted** when combined with a float

Comparison Operators

- Equality: ==
- Inequality: !=
- Greater than: >
- Greater than or equal to: >=
- Less than: <
- Less than or equal to: <=

Logical Operators

- `and`: `True` if both operands are `True`, `False` otherwise

- `or`: `True` if either operand is `True`, `False` otherwise

- `not`: `True` if lone operand is `False`, `False` if it's `True`

Sequence Operators

- Indexing: `seq[index]`
- Slicing: `seq[start:stop]`, `seq[start:stop:step]`
- `in` operator: tests for membership, different from for loop's `in`

Variables and Assignment

- `a = 7`
- `a = a + 1`
- `a += 1`
- Can’t refer to a variable before it’s initialized
  - For loop variables are initialized implicitly
  - So are function parameters

Practice: identify value and type for each expression on the next slide.

- Assume:
  - `x = 1`
  - `s = “swarthmore”`
  - `L = [[7, 29, 0], [-4, 13], [9, 9, 5]]`
- `1 + 2.0`
- `1 / 2`
- `1.0 / 2`
- `21 % 10`
- `s[2:4]`
- `len(L)`
- `len(L[1])`
- `L[1][0] + L[2][2]`

Conditionals / Branching

- `if`
- `if` / `else`
- `if` / `elif` / `elif` / ... / `elif` / `else`
- Nested conditionals

Loops

- `for` loops
  - `for i in range(len(seq)):`
  - `for i in range(n):`
  - `for ch in str:`
  - `for item in lst:`
- `while` loops
  - `while [boolean expression]:`
  - `while True:`

Loops

- Nested loops
- Accumulators
  - Initialize accumulator variable once before loop
  - Update accumulator variable inside loop
  - use final value after loop ends

Built-in functions

- `print(message)`
  - string formatting
- `raw_input(prompt)`
- conversion functions: `int`, `float`, `str`
  - validation with `try/except` or `str.isdigit()`
- `type()` — useful for studying
- `len(seq)`
- `range(start, stop, step)`

Functions from a module/library

- `from random import *`
- `random()`
- `randrange(start, stop, step)`
- `choice(seq)`

Defining Functions

- Arguments in function call match parameters in function definition based on order

- Parameters and variables defined within a function are local to that function.
    - Can't be referred to outside of the function
    - This is what we depict with frames in a stack diagram
- Specifying a return value
    - Or `None` by default

Mutable vs. Immutable

- Strings, ints, floats, and bools are immutable. They can never be modified, only used to compute something new.

- Lists and objects are mutable. They can be mutated in the following ways:
  - index assignment: `L[i] = L[i]+1`
  - calling a method that mutates, like `L.pop(index)` or `L.append(item)`
  - passing a list or object to a function that does mutation

- We use the stack to depict mutability vs. immutability.
  - When we mutate we don't cross out any arrows

Practice: draw stack at line indicated and show all output from program on next slide.

```
def mystery(s, initialCount):
  count = initialCount
  for i in range(len(s)):
    if s[i] in "aeiou":
      s[i] = "_"
      count += 1
  # Draw stack here
  return count

def main():
  myList = ["t", "e", "a"]
  x = 0
  result = mystery(myList, x)
  print(myList)
  print("Result: %d" % result)

main()
```

Recursive Functions

- A recursive function calls itself with a different, somehow smaller set of arguments.

- This is instead of using a loop.

- Recursive functions have a base case and a general case

  - The base case is a very small version of the problem, which can be solved right away.

  - The general case breaks the problem down into a smaller version of the same problem. Uses the solution to the smaller problem in solving the original problem.

Recursive Functions

- Sequences:
  - typical base case: `len(L) == 0` or `len(L) == 1`
  - typical general case: do something with `L[0]`, combine it with result of recursive call on `L[1:]`
- Ints:
  - typical base case: `n == 0` or `n == 1`
  - typical general case: do something with `n`, combine it with result of recursive call on `n-1`
- Take a “leap of faith” when writing the code for the general case. Assume the function will eventually do what you expect, and ask yourself what `f(L[1:])` or `f(n-1)` will return.

Recursion practice

- Write a recursive function, `length(L)`, that returns the length of a list `L` without using the `len` function.

- Write a `main()` function that calls `length(L)` on some list of length 3. Draw the stack diagram as it would look when the base case of the `length(L)` function is reached.

Searching

- Does the value `x` appear in the list `L`?
    - linear search
        - list doesn't need to be sorted
        - O(n) run time
    - binary search
        - list does need to be sorted
        - O(log n) run time
        - Repeatedly cuts in the half the range of indices where `x` might be found
        - Know how `low`, `mid`, and `high` variables update

Sorting

- $O(n^2)$ sorts:
  - selection sort: select item that should go at position i and swap it directly into place.
  - bubble sort: swap consecutive items that are out of order so biggest items “bubble” up
  - insertion sort: insert next item so beginning of list stays sorted
- O(n log n) sort:
  - merge sort: repeatedly split in half until you have lists of size 1, which are already sorted, then merge back together

Practice algorithm traces

- Binary search for 3 in [-4, 1, 3, 7, 8, 10, 12, 17, 20]. Show how low, mid, and high update.
- Trace selection sort on [7, 2, 10, 5, 0]
- Trace bubble sort on [9, 10, 1, 2]

Using classes and methods

- We call constructors to create objects, then call methods on those objects to interact with them
- Strings and lists have methods even though we don't create them by calling a constructor
  - `str.isdigit()`, `lst.append("x")`, `lst.pop(0)`

Syntax for classes and methods

```
# Constructor with no arguments
myList = LinkedList()

# Constructor with arguments
p1 = Point(x, y)

# Method with no arguments
s.isdigit()

# Method with arguments
L.append(27)
```

Defining classes

- A class definition is comprised of method definitions
- The `__init__` method is what actually gets executed when the constructor is called. It initializes the instance variables (`self.whatever`).
- The `__str__` method returns a string representation of the object, usually based on the values of its instance variables. This gets called when the object is printed or converted to a string.
- There are also getter methods, which return the value of an instance variable
- And setter methods, which update the values of one or more instance variables.

Defining classes

- Every method definition must have `self` as its first parameter. This gives the method the ability to access or change the instance variables. When we call a method, we don't list self as an argument.

- We don't know the order in which methods will be called (other than `__init__` being called first) so our methods must make sure that all instance variables are up-to-date before they return

  - i.e. the `turn` method for the bug class had to update `self.bug` and `self.heading`

For the class definition on the next slide:

- Create a `LibraryBook` with title `“Pride and Prejudice”` and author `“Jane Austen”`. Store it in a variable called `book`.

- Show what `print(book)` would display.

- Define a method `checkOut(newBorrower)` which updates whatever instance variables need to be updated after someone new checks the book out.

- Write code to register `“David”` checking the book out and returning it followed by `“Tia”` checking the book out.

```
class LibraryBook(object):

  def __init__(self, title, author):
    self.title = title
    self.author = author
    self.checkedOut = False
    self.borrower = ""
    self.pastBorrowers = []

  def __str__(self):
    s = "%s by %s\n" % (self.title, self.author)
    if self.checkedOut:
      s += "Checked out by %s" % self.borrower
    else:
      s += "Not checked out"
    return s

  def returnBook(self):
    self.checkedOut = False
    self.pastBorrowers.append(self.borrower)
    self.borrower = ""
```

Linked Lists

- Each item in the list corresponds to a node. A node contains the item and a pointer to the next node. We can start at the first node and follow links to all the other nodes. If a node's next point is `None`, we've reached the end.

- `Node` class
  - `self.data`, `self.next`
- LinkedList class
  - `self.head`, `self.tail`, `self.size`

Linked Lists

- Be able to draw the structure, showing `self.head`, `self.tail`, and `self.size`

- Be able to identify various methods: insertions, removals, traversals

- Be able to compare run times of linked list operations and Python list operations.

Linked Lists

	Python list	Linked list
Insert at beginning	O(n)	**O(1)**
Insert at end	O(n)*	**O(1)**
Remove from beginning	O(n)	**O(1)**
Remove from end	**O(1)**	O(n)
Get item at index	**O(1)**	O(n)

Practice: identify in words what each of the following methods does.

```
def method1(self, item):
  newNode = Node(item)
  if self.size == 0:
    self.head = newNode
    self.tail = newNode
  else:
    newNode.setNext(self.head)
    self.head = newNode
  self.size += 1
```

```
def method2(self, item):
  newNode = Node(item)
  if self.size == 0:
    self.head = newNode
    self.tail = newNode
  else:
    self.tail.setNext(newNode)
    self.tail = newNode
  self.size += 1
```

```
def method3(self):
  self.head = self.head.getNext()
  if self.size == 1:
    self.tail = None
  self.size -= 1

def method4(self):
  if self.size == 1:
    self.head = None
    self.tail = None
  else:
    current = self.head
    while current.getNext().getNext() != None:
      current = current.getNext()
    current.setNext(None)
    self.tail = current
  self.size -= 1
```

More Linked List practice

- Write code that creates an empty linked list, stores it in a variable called `LL`, prepends `"A"` and `"B"`, appends `"C"` and `"D"`, and then prints out the linked list.

- Draw the structure of `LL` at this point.

- Write a method `getAtIndex(index)` which returns the item at the specified index in a linked list. Show how you would use it to get the item at index `2` in a linked list called `LL`

Top-down design

- Write the `main()` function first, creating a wish list of functions

- Implement those functions one at a time

- Common functions for our top-down designs:

  - A function that gets and validates user input

  - A function that helps with complicated computation

  - A function that prints output neatly

Write a program to help a hiker setting out on a multi-day hike.
This program should:

- Prompt the user for daily distances hiked.
- Continue asking for daily distances until the user enters a -1.
- Validate input to ensure the user enters integers.
- Store the daily distances in a list.
- Print a table summary with two columns: day and distance.
- The output should be neat and easy to read.
- Print total distance hiked and average distance hiked per day.
- **Have at least two functions besides `main()`** (more if you would like).

Here is an example of the full program (user input in **bold**):

```
Daily miles hiked: 15
Daily miles hiked: 18
Daily miles hiked: twelve
Enter a valid integer!
Daily miles hiked: 12
Daily miles hiked: 0
Daily miles hiked: -1

day  hiked (miles)
 1      15
 2      18
 3      12
 4       0

Total miles hiked over 4 days: 45
Average miles hiked per day: 11.25
```

Practice writing a full program

Write a program to help a hiker setting out on a multi-day hike.
This program should:

- Prompt the user for daily distances hiked.
- Continue asking for daily distances until the user enters a -1.
- Validate input to ensure the user enters integers.
- Store the daily distances in a list.
- Print a table summary with two columns: day and distance.
- The output should be neat and easy to read.
- Print total distance hiked and average distance hiked per day.
- **Have at least two functions besides `main()`** (more if you would like).

Here is an example of the full program (user input in **bold**):

```
1
2 Daily miles hiked: 15
3 Daily miles hiked: 18
4 Daily miles hiked: twelve
5 Enter a valid integer!
6 Daily miles hiked: 12
7 Daily miles hiked: 0
8 Daily miles hiked: -1
9
10 day  hiked (miles)
11  1      15
12  2      18
13  3      12
14  4       0
15
16 Total miles hiked over 4 days: 45
17 Average miles hiked per day: 11.25
```

Good luck!!