Expect questions that ask you to:

- Compute expressions
- Identify the type of an expression
- Write a complete program
- Write a class
- Trace a program, showing output and stack diagram
- Fix bugs in code
- Understand and write recursive functions
Expect questions that ask you to:

• Identify the run time of an algorithm
  • $O(1)$, $O(\log n)$, $O(n)$, $O(n \log n)$, or $O(n^2)$

• Understand searching and sorting algorithms, know their run times

• Understand linked lists
  • Write code that uses them
  • Draw structure
  • Understand, implement, analyze run time of linked list methods
  • Compare linked lists to python lists
Types

- int
- float
- string
- bool
  - True, False
- lists
- None
Expressions and Operators

- Addition / Concatenation: +
- Multiplication / Replication: *
- Subtraction: -
- Division: /
- Integer division
- Modulo/Remainder: %
Expressions and Operators

- Equality: `==`
- Inequality: `!=`
- Greater than / Greater than or equal to: `>`, `>=`
- Less than / Less than or equal to: `<`, `<=`
- and, or, not
- ‘in’ operator
  - Different from for loop’s ‘in’
- indexing (lists and strings)
- slicing (lists and strings)
Variables and Assignment

• $a = 10$

• $a = a + 1$

• $a += 1$

• $*=, -=, /=, %= 1$

• scope of a variable

• can’t refer to a variable before it’s defined
Using built-in functions

- print()
  - string formatting, or at least str() with concatenation
- raw_input()
- int() / float()
  - validation with try/except or .isdigit() method
- type()
- len()
- range(start, end, step)
Using functions from a module

- from random import *
- random()
- randrange()
- choice()
Conditionals / Branching

- if
- if / else
- Multiway conditionals: if / elif / elif / else
  - Can have many elif’s
- Nested conditionals
Loops

• For loops
  • for i in range(len(lst)):
  • for ch in str:
  • for item in lst:

• While loops
  • while [condition that evaluates to True or False]:
  • while True:

• Nested loops

• Accumulators
  • Initialize once before loop
  • Update in loop
  • Print / return / otherwise use final value after loop
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

• Specifying return value
  • Or None by default

• Stack diagrams

• Mutable vs. Immutable
  • Lists and objects can be modified by a function call
  • Strings, ints, floats, Booleans cannot

• Top-down design: splitting a program into functions
Recursive Functions

- Base case, general case

- For general case, break problem into smaller version of same problem

- Lists/strings:
  - Common base case: `len(lst) == 0`
  - Common general case: do something with `lst[0]`, combine it with result of recursive call on `lst[1:]`

- Ints:
  - Common base case: `n == 0` or `n == 1`
  - Common general case: do something with `n`, combine it with result of recursive call on `n-1`

- The “leap of faith”: assume recursive call will work
Using classes/methods

- syntax: object.method(arg1, arg2, …)
- syntax: object.method()

• String methods
  - str.isdigit()
  - str.lower()
  - str.upper()

• List methods
  - lst.append("z")

• Constructor with no params:
  - LL = LinkedList()

• Constructor with params:
  - p1 = Point(50, 50)
Defining classes/methods

- syntax
  - class Node(object)
  - def __init__(self):
  - def __str__(self):
  - def setNext(self, nextNode):
- implicit “self” parameter
- instance variables (self.a = b)

- test code
  - if __name__ == "__main__":
    main()

- get methods, set methods

- Perspective of implementer vs. perspective of client code
Searching

- linear search
  - list doesn’t need to be sorted
  - $O(n)$ or linear run time

- binary search
  - list does need to be sorted (and needs to be a python list, not a linked list)
  - $O(\log n)$ run time

- Know how low, mid, and high update

- Common errors on nasdaq lab:
  - while low < high:  -instead of-  while low <= high:
  - high = len(lst)  -instead of-  high = len(lst) - 1
Sorting

• $O(n^2)$ sorts (slower):
  • selection sort: find item that should go at position $i$
  • bubble sort: swap consecutive items that are out of order
  • insertion sort: sort beginning of list, insert next item so beginning of list stays sorted

• $O(n \log n)$ sort:
  • mergesort: split in half until you have lists of size 1 (already sorted), then merge back together
Linked Lists

• Node class

• LinkedList class
  • self.head, self.tail, self.size

• Draw structure

• Implement and analyze run time of methods
  • Insert, remove, traverse

• Compare with python lists
## Linked Lists

<table>
<thead>
<tr>
<th>Operation</th>
<th>Python List</th>
<th>Linked List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert at beginning (prepend)</td>
<td>O(n)</td>
<td>O(1)</td>
</tr>
<tr>
<td>Remove from beginning</td>
<td>O(n)</td>
<td>O(1)</td>
</tr>
<tr>
<td>Insert at end (append)</td>
<td>O(1) / O(n)</td>
<td>O(1)</td>
</tr>
<tr>
<td>Remove from end</td>
<td>O(1)</td>
<td>O(n)</td>
</tr>
<tr>
<td>Get item at index</td>
<td>O(1)</td>
<td>O(n)</td>
</tr>
</tbody>
</table>
Topics not on final

- vim
- unix
- file i/o
- graphics, turtle graphics
- animation (i.e. sleep() function)
- parallel lists
Topics that would be helpful to know

• exceptions, i.e. try-except

• string formatting

• top down design
Practice Problems - Recursion

- Write a recursive function to:
  - Sum the items in a list
  - Compute the length of a list without using `len()`
  - Count the occurrences of an item in a list
  - Determine if a string is a palindrome
  - Multiply two numbers using `+` but not `*`
Practice Problems - Linked Lists

• Lab 11

• Identify what these methods do:

```python
def unknownMethod1(self):
    count = 0
    curr = self.head
    while curr != None:
        curr = curr.getNext()
        count += 1
    return count

def unknownMethod2(self, item):
    newNode = Node(item)
    if self.size == 0:
        self.head = newNode
        self.tail = newNode
        self.size = 1
    else:
        newNode.setNext(self.head)
        self.head = newNode
        self.size += 1
```
Practice Problems - Complete Program

• Re-do the reading deadlines program from Lab 2, except with some added requirements:
  
  • Don’t ask how many days until the reading is due, instead just continue prompting the user until they have reached their page total
  
  • After the goal is reached, print out a table showing how many pages were read on each day (you should store the pages read in a list to achieve this)

<table>
<thead>
<tr>
<th>Day</th>
<th>Pages Read</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>50</td>
</tr>
<tr>
<td>3</td>
<td>37</td>
</tr>
</tbody>
</table>

• Do validation on the user input

• Use functions
Practice Problems - Writing a class

• Re-do the reading deadlines program again, this time using a class, `ReadingLog`

• The constructor should take the number of pages the student has to read, i.e:

  ```python
  log = ReadingLog(totalPages)
  ```

• You should have these methods:

  • `log.stillReading()`, which returns True if the student still has pages left to read and False once the student has reached their reading goal

  • `log.getDay()`, which returns the current day number. You can use this to print a prompt like:

    ```
    Pages for day 2: 12
    ```

  • `log.read(pages)` which records the number of pages the student read on the current day, and advances to the next day

  • The `__str__` method should return a string that contains the pages read on each day in a table. The table should have two columns like: “Day” and “Pages Read”