MERRIMACK COLLEGE

CSC6013 - Algorithms and Discrete Structures

Python refresher and basic structures - Week 1

Prof. Paulo Fernandes

You're Welcome! Week 1

This is a very fast paced course, so this is the course introduction AND the first course topic: Python refresher and basic structures

MERRIMACK COLLEGE

Agenda Week 1 Presentation

Course Introduction

- Course format
  a. Evaluation
  b. Timeline

Python refresher

- Variables
- Decisions
- Files
- Functions and Parameters
- Loops

Basic data structures

- Arrays
- Linked Lists

MERRIMACK COLLEGE

Course Format

People

Instructor

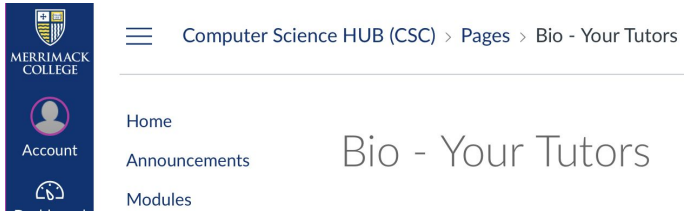
- Prof. Paulo Fernandes
  - fernandesp@merrimack.edu

Success Coach and Graduate Advising

- Use the contact email
  - ecs-grad-advising@merrimack.edu

Student Tutors

- Visit the Hub!



Read the Syllabus!!

Course Format

Live sessions

Read the Syllabus!!

MERRIMACK COLLEGE

8-week long course

Each week typically has:

- A **live weekly class** session **Mondays 6:30-8:30** PM EST via Zoom
  - Meeting ID: 960 3746 8498 Passcode: CSC6013
    - Worksheet assignment during live class with deadline for Friday
    - Project assignment during live class with deadline for next Monday
- **Office hours Wednesdays 8:30-9:30** PM via Zoom
  - Meeting ID: 960 3746 8498 Passcode: CSC6013
- Quiz available Friday, due next Monday

Course Format

Evaluation

You will be evaluated based on

Activity	Percentage
Projects (7)	42%
Quizzes (7)	21%
In-Class Exercises (7)	21%
Final Exam	20%

Late Penalties: All weekly evaluated tasks are due in their stated deadline, the late penalties reduced 10% of the evaluation, plus 2% for each full day of delay

There is a cut-off final deadline at the last day of class (last Friday of the 8th week) - a hard deadline

The hard deadline for the final exam is last week's Sat.

Read the Syllabus!!

Course Format

You will be evaluated based on

The final letter grade is computed according to:

Final grades published first Monday after the course's end.

A	A-	B+	B	B-	C+	C	C-	F
95 and up	90 to 94.9	87 to 89.9	83 to 86.9	80 to 82.9	77 to 79.9	73 to 76.9	70 to 72.9	69.9 and low

Evaluation

Read the Syllabus!!

There are no grade D+, D, or D- in graduate courses.

An average grade B is required to the Masters of Science in Computer Science, lower than that puts you in probation.

Course Format

Timeline

Read the Syllabus!!

MERRIMACK COLLEGE

8-week course

Course Objectives	Week	Topic	Coding Projects	In-class Exercises	Tests
1	1	Python refresher and basic structures	Project #1	Exercise #1	Quiz #1 (unit 1)
1	2	Data structures - Lists, Stacks, Queues, Dictionaries	Project #2	Exercise #2	Quiz #2 (unit 2)
1	3	Data structures - Trees and Graphs	Project #3	Exercise #3	Quiz #3 (unit 3)
2	4	Algorithms - asymptotic notations	Project #4	Exercise #4	Quiz #4 (unit 4)
3	5	Brute force algorithms	Project #5	Exercise #5	Quiz #5 (unit 5)
4	6	Recursive algorithms	Project #6	Exercise #6	Quiz #6 (unit 6)
5	7	Decrease-and-conquer algorithms	Project #7	Exercise #7	Quiz #7 (unit 7)
6	8	Divide-and-conquer algorithms			Final Exam (all units)

Week 1 Python Refresher

We will start with some uses of Python language

To use Python you need to install it on your machine, this can be done in several ways, but probably the safer and easier way is going to www.python.org and just clicking on download! It will download into your machine both the Python interpreter and the IDLE interface.

MERRIMACK COLLEGE

Python 3 and IDLE

You are free to use other IDE (*PyCharm, VScode, etc.*), but in this class we will assume everyone is using IDLE.

Python Language - A new language after version 3

- An interpreted language, unlike Java, C++, etc. that are compiled languages:
  - often a Python program can be called Python script.
- A lot simpler to code
  - Code blocks are defined by indentation;
  - No need to declare variables prior use;
  - Variables can change of type;
  - Simpler input and output;
  - Compact commands.

IDLE - Integrated Development Environment (IDE)

- You can enter a command at a time; or
- Create a Python file (.py) to be interpreted (in real life that is what you should do).

Python Refresher

Basic Python and some...

- **Variables**
- Decisions
- Files
- Functions and Parameters
- Loops

Two Graded Exercises to do

Variables

MERRIMACK COLLEGE

Variables

- **Integer and Real Numbers** (also Complex) - handled alike, but stored differently
  - Variables can be changed from one to another due to operations and casting
- **Strings and Characters** - handled and stored alike
  - Number to String: ***s = str(x)***
  - String to Number: ***x = eval(s)***
- **Booleans** - just ***True*** or ***False***
  - Comparison and Set operations return booleans
- **Lists** - any variable can be deal as a list, an extensible array of variables (indexed access using brackets)
  - Strings a lists of characters
- **User defined** - objects defined using classes (OO)
  - All variables, including user defined objects, can be handled seamlessly

Variables

Dealing with strings

- Basic operations

operator	meaning
+	concatenation
*	repetition
<string>[ ]	indexing
<string>[:]	slicing
len(<string>)	length
for <var> in <string>	iteration through characters

- Built-in methods

.capitalize()	*.title()*	*.replace(<>,<>)*	*.center(#)*
.lower()	*.lstrip()*	*.count(<>)*	*.ljust(#)*
.upper()	*.rstrip()*	*.find(<>)*	*.rjust(#)*

Abundant references on the Internet

String methods

Variables

Strings Indexing

- In Python, a string is a list with all elements being characters
  - Easily accessed by indexing its characters

h1

A	n	t	m	a	n
0	1	2	3	4	5

S	p	i	d	e	r	m	a	n
0	1	2	3	4	5	6	7	8

h2

```
>>> animalHeroes = h1[:3] + " and " + h2[:6] + " Men"
>>> len(animalHeroes)
18
```

A	n	t		a	n	d		S	p	i	d	e	r		M	e	n
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

```
>>> h1 = "Antman"
>>> h2 = "Spiderman"
>>> h1[3]
'm'
>>> h2[5]
'r'
>>> h1[:3]
'Ant'
>>> h2[6:]
'man'
>>> h2[4:8]
'erma'
>>> h1[2:4]
'tm'
>>> len(h1)
6
>>> len(h2)
9
>>> h1[2:4]*3
'tmtmtm'
```

Variables

Abundant references on the Internet

MERRIMACK COLLEGE

[Python Lists] [Python List methods]

Lists - the power of grouping and indexing

- All data types can be turned into lists
  - Lists of characters are **strings**
  - Lists of numbers are **vectors**
  - Lists of variables of the same type are **arrays**
    - Yes, in Python you can have a list where each element may be of different types

+	concatenation
*	replication
.split()	
.reverse()	
.sort()	
.append(<element>	
.remove(element)	
.pop(#)	
.insert(#, <element>)	
.clear()	

User defined variables - Object-Oriented Programming

Variables

- Basic class/object definition
  - One single constructor ***__init__(self,...)***
  - Instance variables ***self.***
  - All methods are public
  - Encapsulation
  - Inheritance
  - Limited polymorphism

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def getName(self):
        return self.name
    def getAge(self):
        return self.age
    def setAge(self,age):
        self.age = age
    def birthday(self):
        self.age += 1

class Student(Person):
    def __init__(self, name, age, major):
        Person.__init__(self,name,age)
        self.major = major

def main():
    p = Person("Chloe", 23)
    s = Student("Beth", 21, "CS")

    print(p.name, "is", str(p.age))
    print(s.name, "is", str(s.age))
```

```
========== RESTART: /Users/pf/Desktop/
Chloe is 23
Beth is 21
>>>
```

Abundant references on the Internet

MERRIMACK COLLEGE

Python Inheritance

Python Refresher

Basic Python and some...

- Variables
- **Decisions**
- Files
- Functions and Parameters
- Loops

Two Graded Exercises to do

Decisions

Abundant references on the Internet

MERRIMACK COLLEGE

Python Conditions

Decisions - the *if* - *elif* - *else* command

- *if* (*condition*):
  - a block of commands to be executed if *condition* is *True*
- *elif* (*condition 2*):
  - a block of commands to be executed if *condition 2* is *True*

  ...
- *else:*
  - a block of commands to be executed if no condition is *True*

Because Python blocks in the code are based on indentation, the *elif* command avoids nested *if*s which would need to be indented.

Decisions

Decisions - the shorthand commands

- *if <condition>: <command>*
  - Saving space.

```
a = eval(input("Enter the first number: "))
b = eval(input("Enter the second number: "))

if (a < b): print("the first is smaller")

if (a > b): print("the second is smaller")

if (a == b): print("they are equal")
```

- *<command> if <condition> else <command>*
  - Used with a single command at a time, but this command can also be nested.

```
a = eval(input("Enter the first number: "))
b = eval(input("Enter the second number: "))

print("the first is smaller") if (a < b) else print("the first is not smaller")

print("the first is smaller") if (a < b) else print("the second is smaller") if (a > b) else print("they are equal")
```

Python Refresher

Basic Python and some...

- Variables
- Decisions
- **Files**
- Functions and Parameters
- Loops

Two Graded Exercises to do

Files

Abundant references on the Internet

MERRIMACK COLLEGE

Python
File Open

Input/Output - file operations

- ***myINfile = open("number.dat", "r")***
  - open a file to read
- ***myOUTfile = open("number.dat", "w")***
  - open a file to write
- ***myfile.close()***
  - close a file
- ***myINfile.read()***
  - it reads all file and stores it in a string
- ***myINfile.readline()***
  - it reads a single line of a file (<end line>) and stores it in a string
- ***print(<expression>,..., file=myfile)***
  - it writes into a file

Files

Abundant references on the Internet

MERRIMACK COLLEGE

Python Try Except

Input/Output - shorthand versions

```
with open("myFile.txt", "r") as infile:
    for line in infile:
        print(line[:-1])
```

```
with open("myFile.txt", "w") as outfile1:
    outfile1.write("my data to write")
```

```
with open("myFile.txt", "w") as outfile2:
    print("my data to write", file=outfile2)
```

- Be aware of possible errors, it might be a good idea to ***try*** and ***except*** file openings

```
try:
    with open("myFile.txt", "r") as infile:
        for line in infile:
            print(line[:-1])
except:
    print("myFile.txt couldn't be opened")
```

Python Refresher

Basic Python and some...

- Variables
- Decisions
- Files
- **Functions and Parameters**
- Loops

Two Graded Exercises to do

Functions and Parameters

Abundant references on the Internet

Python Functions

Functions in Python are defined by *def()*

- A function in Python, unlike typed languages, has no data type associated to it
  - The result of the function (optionally) can be delivered by a ***return*** command
  - The ***return*** command may return as many variables as you want
- Input parameters are defined without type
  - There is no function overload (two functions with the same name, but different parameters), but parameters can have default values
  - The input parameters are passed by value, unless they are complex variables (lists, objects, etc)

Functions and Parameters

Basic function declaration

- A simple function that receives two numeric values and returns the absolute difference between them

```
def absoluteDiff(a, b):
    if (a < b):
        return b-a
    else:
        return a-b
```

```
def absoluteDiff(a, b):
    return ((a-b)**2)**.5
```

- It is called similarly to any other procedural language

```
def main():
    a, b, c = 20, 15, 25
    print("The difference between", a, "and", b, "is", absoluteDiff(a,b))
    print("The difference between", a, "and", c, "is", absoluteDiff(a,c))
    print("The difference between", b, "and", c, "is", absoluteDiff(b,c))

main()
```

Functions and Parameters

Functions with default parameters

```
# default parameters

def desc(a = 1, b = 3, c = 5):
    print("Sum is:", a + b + c)

def main():
    print(" 5 6 7 gives:", end=" ")
    desc(5, 6, 7)
    print(" 5 6 - gives:", end=" ")
    desc(5, 6)
    print(" 5 - 7 gives:", end=" ")
    desc(5, c=7)
    print(" - 6 7 gives:", end=" ")
    desc(b=6, c=7)
    print(" 8 6 7 gives:", end=" ")
    desc(b=6, c=7, a=8)
    print(" - 6 - gives:", end=" ")
    desc(b=6)
    print(" - - - gives:", end=" ")
    desc()

main()
```

- If only some parameters have default values, it should exists from the last to the first
- Produced output:

```
5 6 7 gives: Sum is: 18
5 6 - gives: Sum is: 16
5 - 7 gives: Sum is: 15
- 6 7 gives: Sum is: 14
8 6 7 gives: Sum is: 21
- 6 - gives: Sum is: 12
- - - gives: Sum is: 9
```

Functions and Parameters

Functions return and Python

```
def realRoots(x):
    if (x < 0):
        return "inexistant", "inexistant"
    else:
        return x**0.5, -1*x**0.5

def check(z):
    x, y = realRoots(z)
    if (x == y):
        print("The root of", z, "is", x)
    else:
        print("The roots of", z, "are", x, "and", y)

def main():
    check(16)
    check(-9)
    check(2)

main()
```

```
========== RESTART: /Users/pf/Desktop/roots.py ==========
The roots of 16 are 4.0 and -4.0
The root of -9 is inexistant
The roots of 2 are 1.4142135623730951 and -1.4142135623730951
>>>
```

- Returning more than one variable is equivalent to multiple assignments

Functions and Parameters

Functions with list parameters - implicit output

```
# implicit parameter

# returns the sum of all elements
def sumAll(s):
    acc = 0
    for n in s:
        acc += n
    return acc

# if the sum is negative sort in reverse order,
#   otherwise, sort in non decrescent order
def organize(s):
    s.sort()
    if (sumAll(s) < 0):
        s.reverse()

def main():
    a = [ 12, 7, 28, 31 ]
    print("Before:", a)
    organize(a)
    print("After: ", a)
    b = [ -3, -8, -1, -18 ]
    print("Before:", b)
    organize(b)
    print("After: ", b)

main()
```

- Using lists as input parameter provides implicit output parameters
  - In contrast, the *return* command is an explicit output parameter)

```
Before: [12, 7, 28, 31]
After:  [7, 12, 28, 31]
Before: [-3, -8, -1, -18]
After:  [-1, -3, -8, -18]
```

Python Refresher

Functions and Parameters

- In-Class Exercise E#1.1
  - Function ***zigzag***

Two Graded Exercises to do

Functions and Parameters

Follow strictly the specifications:

- Name of the function;
- Input and output parameters.

In-Class Exercise E#1.1

- Create a function ***zigzag*** that gets three values as input parameters, let's call then ***a***, ***b***, and ***c***
- The program will return True if they are a zigzag, and False otherwise, these numbers are a zig-zag if, and only if ***a*** < ***b*** > ***c*** or ***a*** > ***b*** < ***c***
- For example:
  - If ***a*** = 3 ***b*** = 8 ***c*** = 5 then they are a zig-zag
  - If ***a*** = 3 ***b*** = 8 ***c*** = 9 then they are not a zig-zag
  - If ***a*** = 6 ***b*** = 3 ***c*** = 6 then they are a zig-zag
  - If ***a*** = 3 ***b*** = 5 ***c*** = 5 then they are not a zig-zag

Go to IDLE and try to program it
Save your function in a .py file and submit it in the appropriate delivery room

Deadline: This Friday 11:59 PM EST

Python Refresher

Basic Python and some...

- Variables
- Decisions
- Files
- Functions and Parameters
- **Loops**

Two Graded Exercises to do

Loops

Abundant references on the Internet

MERRIMACK COLLEGE

Python For Loops

Definite Loops - The *for* loop

- Syntaxe:
  - ***for*** *<variable>* ***in*** *<sequence>:*
    <commands>

- The *<variable>* will be of the same type as the elements of the *<sequence>*

```
for i in [0,1,2,3,4]:
    print(i)
```

- The sequence can be expressed as:
  - An explicit sequence
  - A sequence variable
  - A function returning a sequence
    - Usually: ***range****(…)*

```
seq = [0,1,2,3,4]
for i in seq:
    print(i)
```

```
for i in range(5):
    print(i)
```

Loops

Definite Loops - The *range* command

- ***range****(<stop>)*
  - ***range(5)*** is equivalent to [0,1,2,3,4]
  - ***range(11)*** is equivalent to [0,1,2,3,4,5,6,7,8,9,10]
- ***range****(<start>,<stop>)*
  - ***range(1,11)*** is equivalent to [1,2,3,4,5,6,7,8,9,10]
  - ***range(5,15)*** is equivalent to [5,6,7,8,9,10,11,12,13,14]
  - ***range(-2,4)*** is equivalent to [-2,-1,0,1,2,3]
- ***range****(<start>,<stop>,<step>)*
  - ***range(1,10,2)*** is equivalent to [1,3,5,7,9]
  - ***range(1,11,2)*** is equivalent to [1,3,5,7,9]
  - ***range(9,-1,-1)*** is equivalent to [9,8,7,6,5,4,3,2,1,0]
  - ***range(15,3,-3)*** is equivalent to [15,12,9,6]

Loops

An example of definite loops - The strange squares

- The square of the Natural numbers has an interesting property:
  - $1^2 = 1$
  - $2^2 = 1 + 3$
  - $3^2 = 1 + 3 + 5$
  - $4^2 = 1 + 3 + 5 + 7$
  - $5^2 = 1 + 3 + 5 + 7 + 9$

```
# strange square

def sq(x):
    acc = 0
    for i in range(1,2*x,2):
        acc += i
    return acc

def main():
    n = int(input("Enter an int to be squared: "))
    print(n, "squared is", sq(n))

main()
```

How to compute the square of Naturals as such?

```
Enter an int to be squared: 1
1 squared is 1
>>> main()
Enter an int to be squared: 2
2 squared is 4
>>> main()
Enter an int to be squared: 3
3 squared is 9
>>> main()
Enter an int to be squared: 4
4 squared is 16
>>> main()
Enter an int to be squared: 5
5 squared is 25
>>> main()
Enter an int to be squared: 1024
1024 squared is 1048576
```

Loops

Definite loops in specific situations

- A for loop can be used to browse elements of a list

```
mylist = ["A", "B", "C", "D"]
for e in mylist:
    print(e)
```

```
mylist = ["A", "B", "C", "D"]
for i in range(len(mylist)):
    print(mylist[i])
```

  - Both loops print the elements of *mylist*

- A for loop can be used to browse lines of an input file

```
myfile = open(name, "r")
for line in myfile:
    print(line[:5])
```

  - This loop will print the first 5 characters of each line in the inputted *myfile*

Loops

Definite loops the shorthand version

- A for loop can also be used to generate elements of a list:

```
twelves = [12] * 5
```

```
triplet = [eval(input("Number: ")) for _ in range(3)]

twelves = [12 for _ in range(5)]

fromTwelves = [12+i for i in range(5)]

evensFromTwelve = [12+i for i in range(0,10,2)]


print("twelves", twelves, end="\n\n")

print("fromTwelves", fromTwelves, end="\n\n")

print("evensFromTwelve", evensFromTwelve, end="\n\n")
```

```
twelves [12, 12, 12, 12, 12]

fromTwelves [12, 13, 14, 15, 16]

evensFromTwelve [12, 14, 16, 18, 20]
```

Loops

Indefinite loops - The *while* loop

- Syntaxe:
  - ***while*** *<condition>:*
    <commands>

```
i = 0
while (i<5):
    print(i)
    i += 1

for i in range(5):
    print(i)
```

- It executes the commands inside the block while *<condition>* is *True* or until *<condition>* turn out to be *False*
  - It may never execute if *<condition>* is *False* at the first time
  - It may run “forever” if *<condition>* starts as *True* and never changes to *False*

Abundant references on the Internet

Python While Loops

Loops

Breaking out of loops

- The command ***break***
  - It stops the loop and breaks out of it
    - Beware: if used inside nested loops it breaks out the innermost where the command is
  - Usually, it appears inside a decision (***if***) command
- The command ***continue***
  - Similar to break, but it breaks the current loop iteration, but it continues to the next iteration

Loops

Breaking out of loops

```
# ask a number from 1 to 10 and
#   keep on asking until it is a prime number

def getNumber(a,b):
    ans = int(input("Enter an int from "+str(a)+" to "+str(b)+": "))
    while ((ans<a) or (ans>b)):
        ans = int(input("Sorry, it must be an int from "+str(a)+" to "+str(b)+": "))
    return ans

def main():
    primes = [2,3,5,7]
    while(True):
        if (getNumber(1,10) in primes):
            print("Thank you, your number is a prime number")
            break
        else:
            print("Your number is not a prime, please try again.")

main()
```

```
Enter an int from 1 to 10: -32
Sorry, it must be an int from 1 to 10: 13
Sorry, it must be an int from 1 to 10: 6
Your number is not a prime, please try again.
Enter an int from 1 to 10: 7
Thank you, your number is a prime number
```

Python Refresher

Loops

- In-Class Exercise E#1.2
  - Function ***vecSwap***

Two Graded Exercises to do

Loops

In-Class Exercise E#1.2

- Create a function ***vecSwap*** that swaps elements in a created vector
  - Ask the user an even integer between 9 and 21
  - Create a vector sized by this inputted integer
    - [0, 1, 2, …]
  - Swap the first with the second element,
    - Swap the third with the fourth,
      - … and so on
  - Prints out and return the resulting vector

Go to IDLE and try to program it
Save your program in a .py file and submit it in the appropriate delivery room

Deadline: This Friday 11:59 PM EST

Basic Data Structures

Arrays versus Linked Lists

- **Arrays**
- Linked Lists

One Coding Project to do

Arrays

Arrays in Python

- In Python, depending the version, arrays are implemented using Python Lists, which may or may not be actually implemented as regular arrays, with an amount of memory equal to:
  - Number of elements times size of each element
- This amount of memory is indexed by a common arithmetic operation, so data has to be contiguously disposed in the memory
- We will assume this is the way arrays are implemented in Python, as it serves our theoretical analysis needs

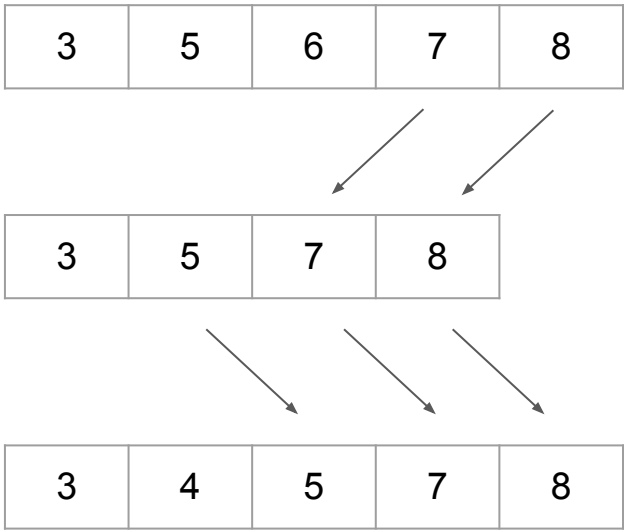
Arrays

Arrays from a Theoretical Point of View

- Arrays are problematic when you need to insert or remove elements
  - To insert or remove an element in an array you need to “scooch over” (copy along) elements

Removing element 6

Inserting element 4 at the second position



Arrays

```
>>> a = [2, 3, 5, 6, 7]
>>> a
[2, 3, 5, 6, 7]
>>> a.pop(0)
2
>>> a
[3, 5, 6, 7]
>>> a.append(8)
>>> a
[3, 5, 6, 7, 8]
>>> a.remove(6)
>>> a
[3, 5, 7, 8]
>>> a.insert(1, 4)
>>> a
[3, 4, 5, 7, 8]
```

Arrays from a Theoretical Point of View

- Yet, in Python the insertion and removal of elements is encapsulated in List methods:
  - *<list>*.***pop***(*<position>*)
    - Remove the element in *<position>*
  - *<list>*.***append***(*<data>*)
    - Insert element *<data>* at the end;
  - *<list>*.***remove***(*<data>*)
    - Remove first instance of element *<data>*;
  - *<list>*.***insert***(*<position>*, *<data>*)
    - Insert element *<data>* in *<position>*;
- Despite being a single command, it implies all "scootch over" kind of operational costs.

Basic Data Structures

Arrays versus Linked Lists

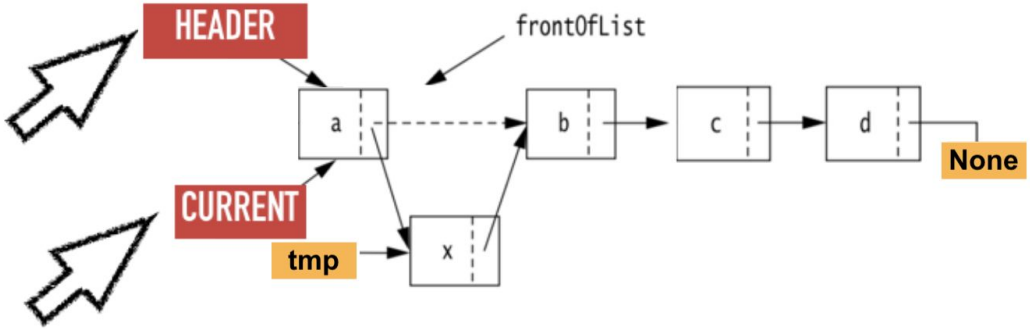
- Arrays
- **Linked Lists**

One Coding Project to do

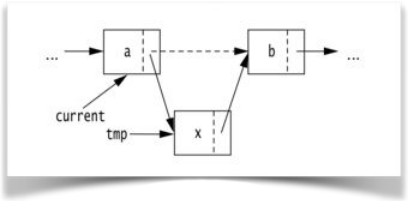
Linked Lists

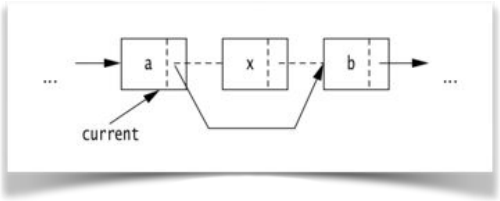
Linked Lists - a trade off compared to arrays

- A structure based on nodes and pointers to other nodes



- Easy to remove and insert





Linked Lists

Linked Lists - Python implementation

- Two classes:
  - The node
    - the data, and
    - a pointer to the next node
  - The linked list
    - a pointer to the first node (header), and
    - a pointer to the current node (current)

```
class Node:
    def __init__(self, d):
        self.Data = d
        self.Next = None
```

```
class LinkedList:
    def __init__(self, d=None):
        if (d == None): # an empty list
            self.Header = None
            self.Current = None
        else:
            self.Header = Node(d)
            self.Current = self.Header
```

None is a Python reserved word for a non-value (similar to Java's NULL)

Linked Lists

Linked Lists - Python implementation

- Inserting a data ***d*** at the Beginning

```
def insertBeginning(self, d):
    if (self.Header is None): # if list is empty
        self.Header = Node(d)
        self.Current = self.Header
    else:                     # if list not empty
        Tmp = Node(d)
        Tmp.Next = self.Header
        self.Header = Tmp
```

```
class Node:
    def __init__(self, d):
        self.Data = d
        self.Next = None
```

```
class LinkedList:
    def __init__(self, d=None):
        if (d == None): # an empty list
            self.Header = None
            self.Current = None
        else:
            self.Header = Node(d)
            self.Current = self.Header
```

Linked Lists

Linked Lists - Python implementation

- Inserting a data ***d*** at the next node of the Current

```
def insertCurrentNext(self, d):
    if (self.Header is None): # if list is empty
        self.Header = Node(d)
        self.Current = self.Header
    else:                          # if list not empty
        Tmp = Node(d)
        Tmp.Next = self.Current.Next
        self.Current.Next = Tmp
```

```
class Node:
    def __init__(self, d):
        self.Data = d
        self.Next = None
```

```
class LinkedList:
    def __init__(self, d=None):
        if (d == None): # an empty list
            self.Header = None
            self.Current = None
        else:
            self.Header = Node(d)
            self.Current = self.Header
```

Linked Lists

Linked Lists - Python implementation

- Removing the node at the Beginning

```
def removeBeginning(self):
    if (self.Header is None): # if list is empty
        return None
    else:                     # if list not empty
        ans = self.Header.Data
        self.Header = self.Header.Next
        self.Current = self.Header
        return ans
```

```
class Node:
    def __init__(self, d):
        self.Data = d
        self.Next = None
```

```
class LinkedList:
    def __init__(self, d=None):
        if (d == None): # an empty list
            self.Header = None
            self.Current = None
        else:
            self.Header = Node(d)
            self.Current = self.Header
```

Linked Lists

Linked Lists - Python implementation

- Removing the node next of the Current

```
def removeCurrentNext(self):
    if (self.Current.Next is None): # if there is no node
        return None                 #        after Current
    else:                           # if there is
        ans = self.Current.Next.Data
        self.Current.Next = self.Current.Next.Next
        return ans
```

```
class Node:
    def __init__(self, d):
        self.Data = d
        self.Next = None
```

```
class LinkedList:
    def __init__(self, d=None):
        if (d == None): # an empty list
            self.Header = None
            self.Current = None
        else:
            self.Header = Node(d)
            self.Current = self.Header
```

Linked Lists

Linked Lists - Python implementation

- Moving the Current

```
def nextCurrent(self):
    if (self.Current.Next is not None):
        self.Current = self.Current.Next
    else:
        self.Current = self.Header
def resetCurrent(self):
    self.Current = self.Header
```

```
class Node:
    def __init__(self, d):
        self.Data = d
        self.Next = None
```

```
class LinkedList:
    def __init__(self, d=None):
        if (d == None): # an empty list
            self.Header = None
            self.Current = None
        else:
            self.Header = Node(d)
            self.Current = self.Header
```

Linked Lists

Linked Lists - Python implementation

- Checking the Current

```
def getCurrent(self):
    if (self.Current is not None):
        return self.Current.Data
    else:
        return None
```

```
class Node:
    def __init__(self, d):
        self.Data = d
        self.Next = None
```

```
class LinkedList:
    def __init__(self, d=None):
        if (d == None): # an empty list
            self.Header = None
            self.Current = None
        else:
            self.Header = Node(d)
            self.Current = self.Header
```

Linked Lists

Linked Lists - Python implementation

- Printing out the list (for demo/debug purposes)

```
def printList(self,msg="====="):
    p = self.Header
    print("====",msg)
    while (p is not None):
        print(p.Data, end=" ")
        p = p.Next
    if (self.Current is not None):
        print("Current:", self.Current.Data)
    else:
        print("Empty Linked List")
    input("----------------")
```

```
class Node:
    def __init__(self, d):
        self.Data = d
        self.Next = None
```

```
class LinkedList:
    def __init__(self, d=None):
        if (d == None): # an empty list
            self.Header = None
            self.Current = None
        else:
            self.Header = Node(d)
            self.Current = self.Header
```

Linked Lists

Linked Lists - Python implementation

Testing (or demo) it all

Full code of Linked Lists demo here

```
def main():
    mylist = LinkedList()
    mylist.printList("List created")
    mylist.insertBeginning(40)
    mylist.printList("Inserting 40 at Beginning")
    mylist.insertBeginning(20)
    mylist.printList("Inserting 20 at Beginning")
    mylist.nextCurrent()
    mylist.printList("Moving the Current to the next (circularly)")
    print("The current is:",mylist.getCurrent())
    mylist.insertCurrentNext(30)
    mylist.printList("Inserting 30 next the Current")
    mylist.nextCurrent()
    mylist.printList("Moving the Current to the next")
    print("The current is:",mylist.getCurrent())
    mylist.resetCurrent()
    mylist.printList("Reseting the Current")
    mylist.insertCurrentNext(25)
    mylist.printList("Inserting 25 next the current")
    print(mylist.removeBeginning())
    mylist.printList("Removing at the Beginning")
    print(mylist.removeCurrentNext())
    mylist.printList("Removing next the Current")
    print("Now, do it again just to be sure you've got it!")

main()
```

Linked Lists

Linked Lists - Python implementation

Testing (or demo) it all

Full code of Linked Lists demo here

```
============= RESTART: /Users/fernandes_paulo/Desktop/linkedlist.py ===
==== List created
Empty Linked List
----------------
==== Inserting 40 at Beginning
40 Current: 40
----------------
==== Inserting 20 at Beginning
20 40 Current: 40
----------------
==== Moving the Current to the next (circularly)
20 40 Current: 20
----------------
The current is: 20
==== Inserting 30 next the Current
20 30 40 Current: 20
----------------
==== Moving the Current to the next
20 30 40 Current: 30
----------------
The current is: 30
==== Reseting the Current
20 30 40 Current: 20
----------------
==== Inserting 25 next the current
20 25 30 40 Current: 20
----------------
20
==== Removing at the Beginning
25 30 40 Current: 25
----------------
30
==== Removing next the Current
25 40 Current: 25
----------------
Now, do it again just to be sure you've got it!
>>>
```

Basic Data Structures

Arrays versus Linked Lists

- Coding Project P#1
  - Array input
  - Linked List creation
  - Linked List handling

One Coding Project to do

First Project

Project #1 - this week's coding assignment

- Create a program that reads a list of Integer numbers from a file named ***data.txt*** (create your own file with about 16 numbers - no repetitions and one number per line)
- Store those numbers into an array ***a*** and sort it
  - ***a.sort()***
- Use the linked list and node classes seen in class to store the ordered elements of ***a*** into a ***LinkedList*** structure ***L***
- Ask the user an Integer value ***x***
- Look for the position to insert ***x*** in ***L***
  - If the value ***x*** is already in ***L***, remove it
  - If it is not, insert ***x*** in the appropriated position so ***L*** remains sorted

First Project

Project #1 - this week's coding assignment

- This program must be your own, do not use someone else's code
- Any specific questions about it, please bring to the Office hours meeting this Wednesday or contact me by email
- This may be a challenging program, and it is intended to make sure you are mastering Python data structure manipulation
- Don't be shy with your questions

Go to IDLE and try to program it
Save your program in a .py file and submit it in the appropriate delivery room

Deadline: Next Monday 11:59 PM EST

That’s all for today folks!

This week’s tasks

- E#1.1 and E#1.2 for the In-Class Exercises
  - Deadline: This Friday 11:59 PM EST
- Q#1 to be available this Friday
  - Deadline: Next Monday 11:59 PM EST
- P#1 coding assignment
  - Deadline: Next Monday 11:59 PM EST

- Try all exercises seen in class and consult the reference sources, as the more you practice, the easier it gets

Next week

- Data Structures
  - Lists
  - Stacks
  - Queues
  - Dictionaries

- Don’t let work pile up!
- Don’t be shy about your questions

MERRIMACK COLLEGE

Have a Great Week!