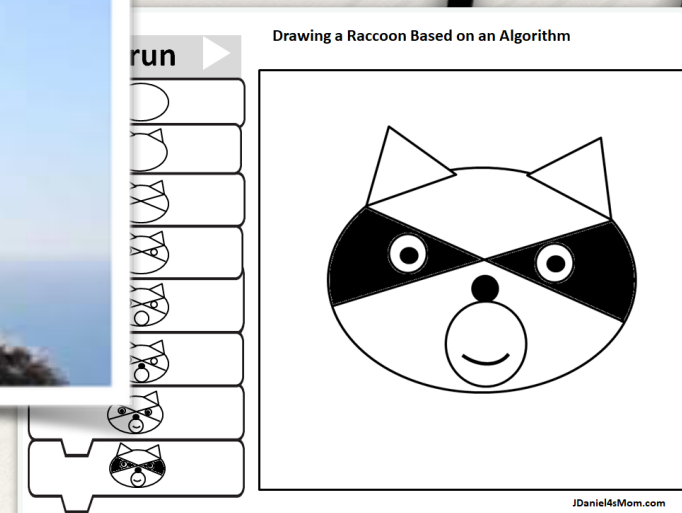


W200 Programming for Data Science

# More about control & algorithms







# Welcome to Week 04

1. Teaching goals for this week
2. Content for the first 8 weeks of the course
3. Observations about assignments and git
4. Pseudocoding, modularity; Working efficiently with breaks.
5. Discussion about loops, iterators, continue & break.
6. Comprehensions.
7. Long breakout room for activities



# 1. Welcome to Week 04

- ❖ This week's teaching goals include good practices for you & your code:
  - ❖ (a) encouraging you to **take a break** when coding [Palomares, 80/20; "sanity checks"].
  - ❖ (b) **breaking down the logic** of your coding - text & drawing - **pseudocoding**. Leads to
    - better functions, better process decomposition and
    - data flows, better object design, and more
    - efficient code.
  - ❖ (c) thinking about **loops & iterators** - when to use one over another; getting comfortable with **comprehensions**.



# 2. First 8 Weeks - Programming

Unit 1 | Introduction, the Command Line, Source Control

Unit 2 | Starting Out with Python

Unit 3 | Sequence Types and Dictionaries

Unit 4 | More About Control and Algorithms

Unit 5 | Functions

Unit 6 | Modules and Packages

Unit 7 | Classes

Unit 8 | Object-Oriented Programming

*Some of the main points include*  
(1) iterators vs. loop;  
(2) the importance of  
pseudocoding, and  
(3) planning ahead for  
algorithm efficiency.



### 3. Helpful Git Term | “git stash”

- If you’ve modified files in a repository but want to keep them without “committing” them, try “git stash” to stash the changes before using “git pull” or “git add”.
- Stash states away untracked changes.
- See <https://gist.github.com/carnivore/997001>
- <https://git-scm.com/downloads>



# 3. Observations about HW



- ❖ Grading questions: Please email all instructors so we can discuss.
  - ❖ Before submitting code, test; check any math/formulae.
- 
- ❖ Formatting: Use string formatting to print without gaps and with a logical number of digits. You don't (usually) need to separate the formatting by a comma:  

```
print("The total including tip would be ${:,.2f}".format(total))  
print("The total including tip would be", "${:,.2f}".format(total))
```



# 3. Observations about HW

- ❖ The **user experience** matters - in coding and in data analysis
- ❖ Consider **number-based** inputs vs. text based. Sometimes it makes sense to number the options and ask the user for an integer.
- ❖ `.lower()`, `rstrip()`, `strip()` can be useful on input to ensure a standard string format. Try to control user (and file) input:  
“**error-correction on input.**”
- ❖ You can use a while loop for data validation.
- ❖ You can “update” a variable each pass through a loop to build strings, lists, etc. over time. Even though strings aren’t mutable. You’re creating a new string with the same variable name each pass through the loop.



# 4. Pseudocoding & Algorithms

## What is Pseudocoding?

“Pseudocode is an informal high-level description of the operating principle of a computer program or other algorithm. It uses the structural conventions of a normal programming language, but is intended for human reading rather than machine reading.” (Wikipedia)

Planning leads to smart coding!

Modularization.

Saves time, and in the real world, saves money.

*Practice, practice, practice this functional decomposition exercise.*

*[https://en.wikipedia.org/wiki/Software\\_design](https://en.wikipedia.org/wiki/Software_design)*



# 4. Pseudocoding: Example

This program calculates the Lowest Common multiple for excessively long input values

```
function lcmNaive(Argument one, Argument two){  
    Calculate the lowest common variable of Argument  
    1 and Argument 2 by dividing their product by their  
    Greatest common divisor product  
  
    return lowest common multiple  
end  
}  
function greatestCommonDivisor(Argument one, Argument two){  
    if Argument two is equal to zero  
        then return Argument one  
  
    return the greatest common divisor  
end  
}  
{  
In the main function  
  
print prompt "Input two numbers"  
  
Take the first number from the user  
Take the second number from the user  
  
Send the first number and second number  
to the lcmNaive function and print  
the result to the user  
}
```

Pseudocode (top)  
Java code (ight)

```
// This program calculates the Lowest Common multiple  
// for excessively long input values
```

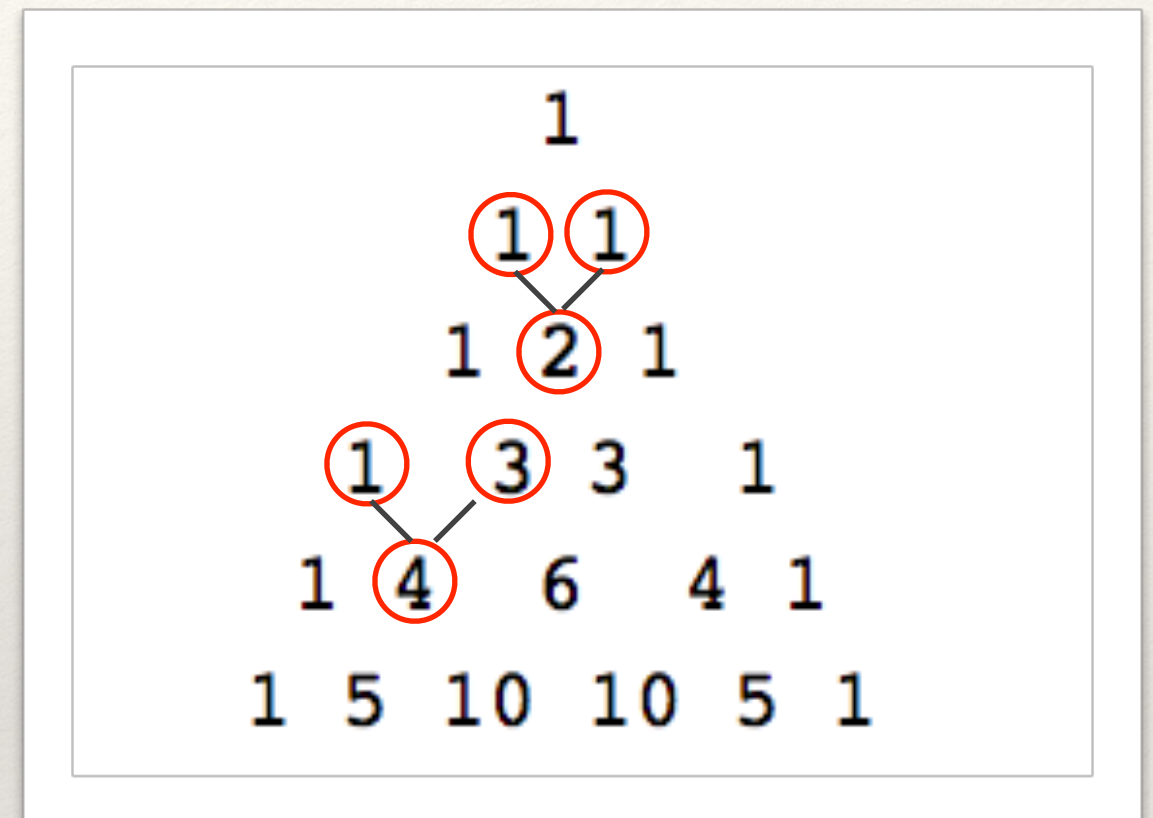
```
import java.util.*;  
  
public class LowestCommonMultiple {  
  
    private static long  
    lcmNaive(long numberOne, long numberTwo)  
    {  
  
        long lowestCommonMultiple;  
  
        lowestCommonMultiple  
            = (numberOne * numberTwo)  
            / greatestCommonDivisor(numberOne, numberTwo);  
  
        return lowestCommonMultiple;  
    }  
  
    private static long  
    greatestCommonDivisor(long numberOne, long numberTwo)  
    {  
  
        if (numberTwo == 0)  
            return numberOne;  
  
        return greatestCommonDivisor(numberTwo, numberOne % numberTwo);  
    }  
  
    public static void main(String args[])  
    {  
  
        Scanner scanner = new Scanner(System.in);  
        System.out.println("Enter the inputs");  
        long numberOne = scanner.nextInt();  
        long numberTwo = scanner.nextInt();  
  
        System.out.println(lcmNaive(numberOne, numberTwo));  
    }  
}
```



## 4. Pseudocoding: Pascal's Triangle

### Pause - Draw it out

- ❖ These statements should be distilled to simplicity
- ❖ There are 1s flanking each row
- ❖ Adding each possible pair of previous row makes next row without the flanking 1s





# 4. Pseudocoding & Algorithms

We will use two arrays, one for the "last row" one for "current row"

Insight: Each pass through the loop, we can set "current row" to "last row". Then we can build a new "current row".

First initialize the two arrays to [1] and [1,1] for rows 1 and 2

We then loop through an algorithm as follows:

1. Replace "last\_row" with the contents of "current\_row" so we can create a new "current\_row".
2. Clear out "current\_row" to have nothing in it.
3. The 0 digit in current\_row is 1
4. Start a "While" loop that iterates until `len(current_row) == len(last_row)`
  5. Use a counter (n) that starts at 1 and increments by 1 each time through the loop
  6. Use "append" to set `current_row[n] = last_row[n] + last_row[n-1]`
7. After exiting the loop, add the final "1"

! Now wait, what is the problem? This only builds one row. Go into breakouts and, starting with this pseudocode, update the pseudocode to find the "N"th line, as required in the problem.



# 4. Pseudocoding & Algorithms

---

What is the underlying purpose of teaching search algorithms?

**Answer:** To show that creative code solutions can drastically improve runtimes.

As you take courses on data at scale, it's not enough for your code to work. It has to work, efficiently. Planning and pseudocoding help

Unit 6 will explore this further.



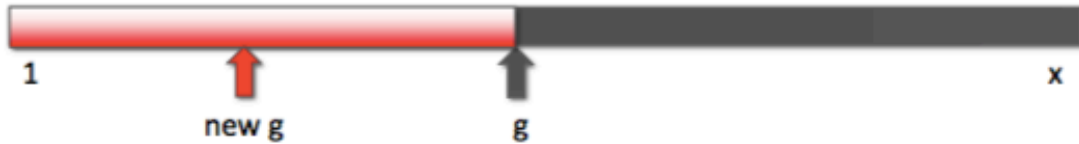
# 4. Pseudocoding & Algorithms

Discuss: What is the difference between the three methods?


- ❖ Brute force
- ❖ Bisection search
- ❖ Heron's method

**BISECTION SEARCH**

- If not close enough, is guess too big or too small?
- If  $g^2 > x$ , then know  $g$  is too big; but now search



- And if, for example, this new  $g$  is such that  $g^2 < x$ , then know too small; so now search



- At each stage, reduce range of values to search by half

6.00.1X LECTURE 15



## 5. Iterating through data

- ❖ Here are some discussion topics, followed by examples. We'll conclude with a breakout room activity where you apply them all.
  1. What's the difference between a for loop and a while loop?
  2. Is one more general than the other?
  3. Why might you want to exit a loop early?
    1. What's the benefit of “break”, of “continue” and of “else” after a loop?
    2. What's a boolean flag? When to use one?



## 5. Looping | For vs. While

Question: What is the difference between a “for” loop and a “while” loop?

Answer: A “while” loop iterates until a condition is met.

The programmer may not know in advance how many iterations are needed.

A “for” loop iterates over a predefined group of objects, acting once on each one.

Generally, use a “for” loop when you have a known set of items to iterate over. Use a “while” loop when you are not sure how many iterations are needed in advance.

All “for” loops can be re-written as “while” loops. In fact, in the first weeks of this course we had you practice implementing several items as “while” loops that are easier to implement as “for” loops.



## 5. Looping | Examples

```
countdown = 5
while countdown > 0:
    print(countdown)
    countdown -= 1
print("Blast off!")
```

```
5
4
3
2
1
Blast off!
```

```
x = ["Paul", "Bill", "Kay"]
for name in x:
    print(name + ", as himself")
```

```
Paul, as himself
Bill, as himself
Kay, as himself
```



# 5. Exiting Loops | *Three Tools*

Question: Why might we want to exit a loop early?

1. Why “break” a loop?
2. Why use “continue”?
3. Why use “else” after a loop?



# 5. Exiting Loops | *Explanations*

**Question: Why might we want to exit a loop early?**

Break exits the entire loop early, preventing our code from running needlessly.

Continue skips the rest of the iteration, and starts the next part of the loop. There is no need to continue processing a row.

Else can be used after a loop completes, and triggers if the loop never “breaks”.



# 5. Exiting Loops | *Break*

When would we use “break”?

In the prime checker code, we do not need to check any values after we find the first non-prime value.

❖ \* What is a boolean flag?

```
x = int(input("Enter a number:"))
prime = True
for i in range(2,x):
    print("checking potential factor:",i)
    if x % i == 0:
        prime = False
        break
if prime == True:
    print(x,"is prime.")
else:
    print(x,"is not prime.")
```

```
Enter a number:25
checking potential factor: 2
checking potential factor: 3
checking potential factor: 4
checking potential factor: 5
25 is not prime.
```



# 5. Exiting Loops | *Continue*

Question: When would we use “continue”?

Use of “continue” early in a loop can save you processing time on many iterations, or when only some values need to be processed (e.g., vowels).

```
word = input("Enter a word: ").lower()
last = "a"

for letter in word:
    if letter not in "aeiou":
        continue
    if letter < last:
        print("The vowels in", word, "are not ordered.")
        break
    last = letter
```

**See whether the vowels in the input are ordered**



# 5. Exiting Loops | *Break/Else*

Question: When would we use “else”?

“Else” can be used to save coding effort by processing the data in a certain way if your entire loop runs.

```
word = input("Enter a word: ").lower()
last = "a"

for letter in word:
    if letter not in "aeiou":
        continue
    if letter < last:
        print("The vowels in", word, "are not ordered.")
        break
    last = letter
else:
    print("The vowels in", word, "are ordered.")
```

```
Enter a word: Pual
The vowels in pual are not ordered.
```



## 6. Comprehensions | *Purpose*

---

Question: What is a “list comprehension”?

Answer: A comprehension is a single-line “for loop” that enables you to quickly build a list. You can also build sets or dictionaries.

Question: Why would we want to use one?

Answer: They are fast, efficient, and fun ways to build sequences out of other sequences!

Notice that we build more complicated conditions - so note the logic and the syntax as we progress.



## 6. List Comprehensions | a compact for loop

List comprehension implicitly appends items resulting from the “Expression”

**Structure : [Expression(item) *for* Item *in* Iterable]**

**For loop:**

```
squares = []
for i in range(1,11):
    squares.append(i**2)
print(squares)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

**List comp:**

```
[i**2 for i in range(1,11)]
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```



# List Comprehensions | *loops with conditionals*

List comprehension is run on items also in the second iterable

**Structure : [Expression(item) *for* Item *in* Iterable *if* Item *in* container]**

```
import string

word = '2345-Nessie ate-YOU_TOO!!@#'.lower()

[i + ' vowel' for i in 'aeiou' if i in word]

['a vowel', 'e vowel', 'i vowel', 'o vowel', 'u vowel']
```



# List Comprehensions | *loops + loops*

List comprehension produces tuples resulting from NESTED loops

**Structure :** [(**Expr1(item1)**, Expr2(item2)) *for Item1 in Iter1 for Item2 in Iter2*]

```
[(row, column) for row in range(4) for column in range(4)]
```

```
[(0, 0),  
 (0, 1),  
 (0, 2),
```

[(0,0),(0,1),(0,2),(0,3),(0,0),(1,0),(1,1),(1,2),(1,3)...(3,3)]

Try this on [pythontutor.org](http://pythontutor.com/visualize.html#togetherjs=6mOAavcRgK) <http://pythontutor.com/visualize.html#togetherjs=6mOAavcRgK>



# Other Comprehensions | *Key:value pairs*

Curly {} braces and key value pairs define this one

**Structure : {Item : Expression(item) for Item in Iterable}**

```
text = "Here is some example text."
frequencies = {letter : text.count(letter) for letter in text}
print(frequencies)
```

```
{'t': 2, 'o': 1, 'e': 6, 'l': 1, 'H': 1, 'i': 1, 'r': 1, 'x': 2, '.': 1, 'm': 2, 'p': 1, ' ': 4, 'a': 1, 's': 2}
```



# 7. Activities

Activity 1 - Finish the **pseudocode** then implement it.

Work on the other activities, especially the .json parts.

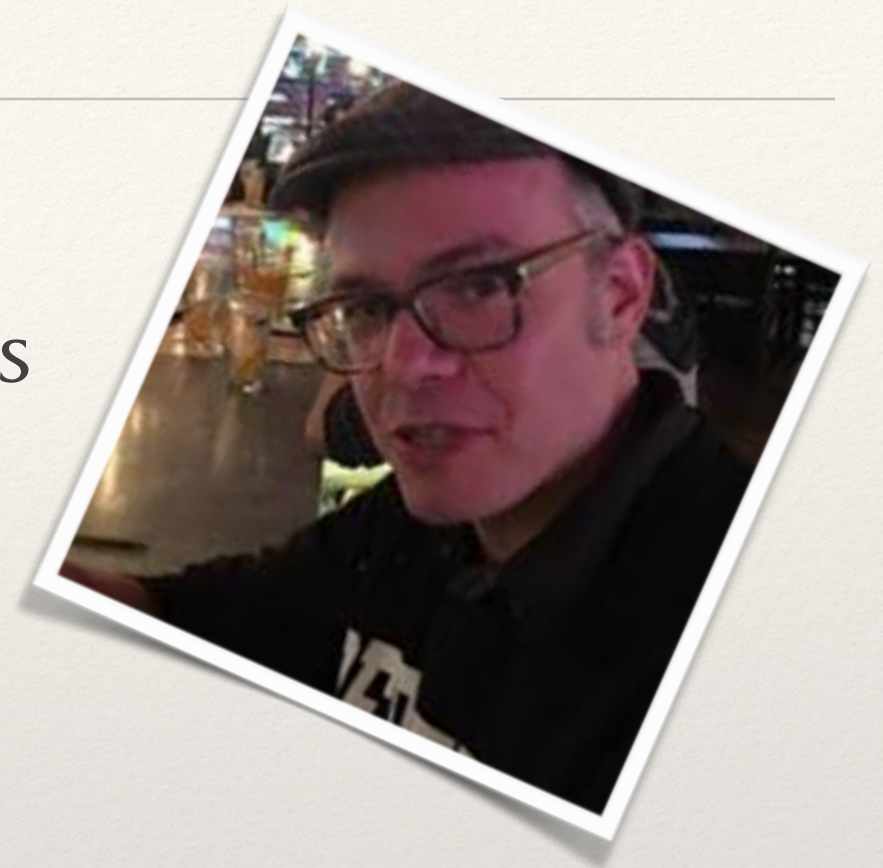
Fun Fact: Your entire Jupyter Notebook is actually saved as JSON. You will see this if you ever need to resolve a Jupyter “merge conflict”.

```
,
{
  "cell_type": "code",
  "execution_count": 36,
  "metadata": {
    "collapsed": true
  },
  "outputs": [],
  "source": [
    "### notes for HW2 ###"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "collapsed": true
  },
  "outputs": [],
  "source": [
    "# averages UI issues\n",
    "# from a UI standpoint you need to tel\n",
    "# acceptable\n",
    "\n",
    "# you could have imported math.sqrt (n\n",
    ]
},
```



# Recap:

- ❖ for and while loops
- ❖ breaking and continuing in nested if-statements
- ❖ comprehensions
  - ❖ (soon we'll look at lambda functions that'll build on this idea)
- ❖ functional decomposition/data flow - pseudocoding & modularity
  - ❖ (check the optional project management document in resources)



*Start thinking about  
your projects  
and practice  
pseudocoding.*