Reminders

• Regrade Requests-9/28-9/30 via Gradescope

• Assignments
  • APT 2 due today
  • APT 3 live today
Key instructions

• Input
• Output
• Assignments* ✔
• Math/Logic ✔
• Conditionals ✔
• Repetition

*not listed in book
Python Data Types

• int, float, bool ✔

• Collections
  • Strings ✔
  • Lists ✔
  • Tuples
  • Sets
  • Dictionaries
PFTD

• Loops
  • Turtles
• Accumulator
  • Bagels
• Files

“The mere imparting of information is not education.”
  • Dr. Carter G. Woodson
KISS Principle

- Think of the non-computing context for any word/terms
- KISS model
  - Work smarter, not harder!!
- “Good programmers are simply good designers.”
  - Dr. Washington
- Design first and always!
- Importance of reusability
- USE PYTHONTUTOR IF YOU HAVE QUESTIONS!
People to Know: Prof. Victoria Chávez

- BA-Computer Science & Hispanic Literatures and Culture (Brown)
- MA-Urban Education Policy (Brown)
- SNAPy creator
  - SMS-based app, where food stamps accepted
- Lecturer
  - University of Rhode Island (CS)
- Consultant
  - CS for Rhode Island (CS4RI)
Turtle Programming

• Must:
  • Import turtle module
  • Create window/Screen and exit on click
  • Create turtles to use, name/type/value

• Review Turtle commands and concepts

• See Snowpeople.py, ColorMyWorld.py, and Spiro.py for some ideas (zip file linked in course calendar)
  • Color, Position, Leaving Turtle where started
  • Many more commands than this
What are key concepts in Spiro.py?

```python
import turtle

def draw(turt):
    colors = ['red', 'purple', 'blue', 'green', 'yellow', 'orange']
    turt.speed(0)
    for x in range(360):
        turt.pencolor(colors[x % 6])
        turt.width(x/100 + 1)
        turt.forward(x)
        turt.left(59)

if __name__ == '__main__':
    win = turtle.Screen()
    t = turtle.Turtle()
    draw(t)
    win.exitonclick()
```

- **Import turtle**
- **Create screen/window**
- **Create turtle**
- **Close on click**
- 1 – slowest
- 10 – fastest
- 0 – No animation
Question

After the code executes, where is the turtle with respect to where it started?

```python
if __name__ == '__main__':
    win = turtle.Screen()
    t = turtle.Turtle()
    t.forward(100)
    t.backward(50)
    t.forward(-50)
    win.onclick()
```
Activity 1:

- Equilateral triangle
  - Corner degrees: 60
  - Side length: 50
- Demo: PyCharm
Useful turtle functions

- `forward(n)/backward(n)` – move turtle n pixels
- `left(n)/right(n)` – turn turtle n degrees
- `pendown()/pendup()` – whether actually drawing
- `setposition(x, y)` – puts turtle in this (x,y) coordinate (a.k.a. goto, setpos)
- `sethead(n)` – points turtle in this direction (n=0 is east)
- Many more in documentation!
  - [https://docs.python.org/3/library/turtle.html](https://docs.python.org/3/library/turtle.html)
Turtle Concepts

• Create a screen so you can ..
  • Exit On Click
  • Some other Screen Functions
• Create a turtle so you can ...
  • Move and draw using the turtle
• Drawing Concepts
  • Pen [up and down]
  • Fill
  • Color
  • Position
Bagels
(Accumulation)
APT Bagels

• How figure out how many bagels needed?
  • 7-steps!
Step 1 and 2

• Step 1: Solve an instance (TPS)
  • orders = \[11, 3, 24, 17\]
Step 1 and 2

• Step 1: Solve an instance (TPS)
  • orders = [11, 3, 24, 17]
  • Total: 58

• Step 2: What did we do?
  • $11 + 3 + (24+2) + (17+1) = 58$
Step 3: Generalize

- Go through list
- If less than 12
  - Do nothing
- If greater than or equal to 12
  - Add however many times 12 goes into the order
- Sum everything
Step 4: Test steps

- Go through list
- If less than 12
  - Do nothing
- If greater than or equal to 12
  - Add however many times 12 goes into the order
- Sum everything

- [11, 22, 33, 44, 55]
- 11
  - Nothing (less than 12)
- 22
  - +1
- 33
  - +2
- 44
  - +3
- 55
  - +4
- Sum: 175
Step 5: Code

- Go through list
- If less than 12
  - Do nothing
- If greater than or equal to 12
  - Add however many times 12 goes into the order
- Sum everything

- for loop!
- if statement
- if’s or if…else statement?
  - floor div: //

Could we use the accumulator pattern? Yes!
Step 5: Code

```python
def bagelCount(orders):
    """
    return number of bagels needed to fulfill the orders in integer list parameter orders
    """
    total = 0
    for order in orders:
        if order < 12:
            total += order
        else:
            extra = order // 12
            total += order + extra
    return total
```

- **Initialize before loop**: `total = 0`
- **Update inside loop**: `total += order`
- **Do something with value after loop**: `return total`
Code-Tracing a Loop

1. Find the changing variables/expressions
2. Create table, columns are variables/expressions
   1. First column is loop variable
   2. Add columns to help track everything else
3. Each row is an iteration of the loop
   1. Before execute code block, copy down each variable’s value
   2. Execute code block, update a value in the row as it changes
Code-Tracing a Loop

1. Find the changing variables/expressions
2. Create table, columns are variables/expressions
   1. First column is loop variable
   2. Add columns to help track everything else

```python
def mystery(lst):
    idxMax = 0
    for i in range(len(lst)):
        if lst[idxMax] < lst[i]:
            idxMax = i
    return idxMax
```

TPS: What should be the table’s columns?
Code-Tracing a Loop

1. Find the changing variables
2. Create table, columns are the variables
   1. First column is loop variable
   2. Add columns to help track everything else

```python
def mystery(lst):
    idxMax = 0
    for i in range(len(lst)):
        if lst[idxMax] < lst[i]:
            idxMax = i
    return idxMax
```
1. Before execute code block, copy down each variable's value

2. Execute code block, update a value in the row as it changes

```python
def mystery(lst):
    idxMax = 0
    for i in range(len(lst)):
        if lst[idxMax] < lst[i]:
            idxMax = i
    return idxMax

mystery([2, 12, 4, 15, 15])
```
TPS: Fill in table

1. Before execute code block, copy down each variable’s value
2. Execute code block, update a value in the row as it changes

```python
def mystery(lst):
    idxMax = 0
    for i in range(len(lst)):
        if lst[idxMax] < lst[i]:
            idxMax = i
    return idxMax
```

mystery([2, 12, 4, 15, 15])

<table>
<thead>
<tr>
<th>i</th>
<th>idxMax</th>
<th>lst[idxMax]</th>
<th>lst[i]</th>
<th>lst[idxMax] &lt; lst[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>False</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
TPS: Fill in table

1. Before execute code block, copy down each variable’s value

2. Execute code block, update a value in the row as it changes

```
def mystery(lst):
    idxMax = 0
    for i in range(len(lst)):
        if lst[idxMax] < lst[i]:
            idxMax = i
    return idxMax

mystery([2, 12, 4, 15, 15])
```

<table>
<thead>
<tr>
<th>i</th>
<th>idxMax</th>
<th>lst[idxMax]</th>
<th>lst[i]</th>
<th>lst[idxMax] &lt; lst[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>False</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>True</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>12</td>
<td>False</td>
</tr>
</tbody>
</table>
Examples of Processing Data

• Google Ngram viewer
  • https://books.google.com/ngrams
Studying Language Evolution

• Ngram informs how words evolve
• From dove to dived
• https://www.youtube.com/watch?v=tFW7orQsBuo
Sequences, Repetition

• Parameters? What are they to this query?

What can the URL tell you?
Sequences, Repetition

- Parameters? What are they to this query?

Processing Data

• How do we find the longest word in .. Any text?
• How do we find the word that occurs the most?
• How is this related to how Google Search works?

• Text files can be viewed as sequences
  • Sequences of lines
  • Each line is a string
  • Some clean-up because of ‘\n’
One line at a time

• Simplest and reasonably efficient Python pattern
  • Open, loop, close, return/process
  • LineCounter.py in code zip file

• File as sequence
  • One line at-a-time

• Asymmetry in Open vs Close steps
File Objects

• A file is an object, like a string
  • Functions applied to object: \texttt{len(“word“)}
  • To get file object use \texttt{open(“data.txt“)}
  • What is returned? Integer value, file object

• Often methods (aka function) applied to object
  • \texttt{f.readlines()}, \texttt{f.read()}, \texttt{f.close()}
  • Just like: \texttt{st.lower()}, \texttt{st.count(“e“)}
Text File Processing Pattern

• See module `FileStuff.py` in code zip file
  • If newline `\n` is read, call `.strip()`
  • If want to break line into “words”, call `.split()`

• Process the list returned by `.split()`
  • May need to convert strings to int or float or …

• The `for line in f:` pattern is efficient
  • Contrast list returned by `f.readlines()`
Lists of Data

- String lists: ["ant", "fox", "cat", "dog"]
- Lists of int/float numbers: [5, 3.14159, -15]
- What about lists of lists? Variable plist = [["Washington", 1789, 57], ["Clinton", 1993, 46]]

- What is plist[0]?
- What is plist[0][2]?
- Can always use a variable:
  • val = plist[0], then val[2]
Reminders

• Work smarter, not harder
• Design first
• Try to identify where you are stuck
  • Identify resources to help solve problem
• Leverage your design and PythonTutor to understand program flow of control
  • http://pythontutor.com