Topic 1b - Programming Difficulties (Part 2, Small Group Consolidation)

Uninitialized variable

- Traceback (most recent call last):
  File "/Users/schafer/Desktop/python_code/temp.py", line 3, in <module>
    sum = sum + value

  Traceback (most recent call last):
  File "<pyshell#0>" line 1 in <module>
  total = total + value
  NameError: name 'total' is not defined
  >>> |

- TypeError: unsupported operand type(s) for +: 'builtin_function_or_method' and 'int'

Cause(s)

- A variable on the right-hand-side of an assignment statement was not initialized, sum in this case.

Fix(es)

- Be sure to initialize all accumulator variables, e.g.,
  sum = 0.
- Remember that the input() function produces a string value—convert it to the appropriate numeric type, e.g.,
  value = int(input("Enter a value")).

Misplaced initialization

- The accumulator variable doesn't get all the values

Cause(s)

- The initialize statement is placed inside the loop that accumulates the values, e.g.

  ```python
  count = int(input("How many values?"))
  for index in range(count):
    sum = 0
    value = int(input("value?"))
    sum = sum + value
  print( str(sum) )
  ```

Fix(es)

- Move the initialization outside (before) the loop so it only occurs once, e.g.,

  ```python
  count = int(input("How many values?"))
  sum = 0
  for index in range(count):
    value = int(input("value?"))
    sum = sum + value
  print( str(sum) )
  ```
Undefined Variable

- When you attempt to use something as a variable that has not been set.

Cause(s)

- This error happens when python is trying to find a name in the text and there is no memory of that text available.

```python
tigh = 10
bakker = 15
tighBakker = tigh * bakker
print(tighBakker)
```

Fix(es)

- Make sure you have all of your code typed correctly and that all of your calculations are lined up with current names.

Coding More Efficiently

- When you get your code to work as intended, but you know there was a better way to do it that you couldn't think of yourself.

Cause(s)

- Wanting to finish a project at all cost, even if it means sacrificing efficiency.
- Below you will see my long version of a coding followed by the MUCH shorter version.

```python
def seedWins(infile):
    allData = matchup(infile)
    one = 0
    two = 0

    sixteen = 0
    total = len(allData)

    gamesWon = []

    for i in range(len(allData)):
        game = allData[i]
            one = one + 1
            two = two + 1

    sixteen = sixteen + 1

    print("The number of NCAA Tournament wins by seed from 1985 to 2019 are listed below: ")
    print("1 seed = " +str(one)+" wins.")
    print("2 seed = " +str(two)+" wins.")

    print("15 seed = " +str(fifteen)+" wins.")
    print("16 seed = " +str(sixteen)+" wins.")
```

Compared to:
Fix(es)
- Practice, experience, and ask questions of your peers and instructors!

Using the Correct Type of Number
Error Message:
Traceback (most recent call last):
  File "/Users/mwilder/Desktop/UNI Comp Science/FOP/Unit 2/6 a,b,c,d/corn.py", line 2, in <module>
    volume = int(volume)
ValueError: invalid literal for int() with base 10: '12.2'

Cause(s)
- In the code the volume was entered as an int, rather than a float. When the user tries to input a number with a decimal, it is not read as a valid command.

Fix(es)
- Instead of int(volume), it should be typed as float(volume)

Parallelism Bugs
- Some novice programmers may assume that computers can execute (or even consider) multiple lines of codes in parallel although it runs each code line-by-line from top to bottom (called parallelism bugs)

Cause(s)
- Putting things out of order. In the below example, we can interpret what should happen, but the computer will execute “top-down”.

```python
def seedWins(inFile):
    allData = matchup(inFile)

    #Setup a length 16 list with each value = 0
    wins=[]
    for x in range(16):
        wins.append(0)

    #run through data
    for game in allData:
        #who won
        if (game[2]>game[3]):
            seed=game[0]
        else:
            seed=game[1]

        index=seed-1  #Seeds are 1-16 but we need 0-15 for the list
        wins[index] += 1

    print("The number of NCAA Tournament wins by seed from 1985 to 2019 are listed below: ")

    for index in range(16):
        seed=index+1
        print(str(seed)+"seed = "+str(wins[index])+" wins.")
```
\[
\begin{align*}
\text{AREA} &= H \times W \\
H &= 10 \\
W &= 5 \\
\text{PRINT AREA}
\end{align*}
\]

Fix(es)
- Define the “H” and “W” variables before “AREA”.

**Intentionality Bugs**
- Novices may assume that computers can do what programmers intend, but is not explicitly instructed.

**Cause(s)**
- Occurs when a novice read program codes while considering a solution plan.
- Example: student wanted to repeat a set of commands three times using a “while loop” that included a counter having an initial value that was zero. The counter was increased by one per iteration. The actual code said “repeat while counter is less than or equal to 3”. He thought the loop would stop after three iterations and the counter would be three. His plan was correct and the counter was traced correctly. However, computers would run another iteration because the while loop ran until the condition got false rather than reach 3 unlike the student’s intention.

Fix(es)
- Use less than instead of less than or equal to.
- This error pops up in other similar situations.

**Egocentrism Bugs**
- Novices may assume that computers can do what programmers mean, but is not explicitly instructed.

**Cause(s)**
- Occur when novices omit necessary specifications in an assumption that computers will know what they mean.
- Example: novices often use a same name for two different variables because they know the different roles of them while computers have no instruction about it.
- Also occur when novices design program only from their perspective.
  - Example: fourth-grade students interpreted the direction of moves and turns based on their orientation reflected on a screen rather than a turtle’s orientation in the programming language Logo.

Fix(es)
- Use unique variable names.
Other Notable Difficulties While Programming to Consider

- Understanding how to design a program to solve a certain task.
  - Difficult to apply learned concepts onto a larger and/or more difficult scale.
- Dividing functionality into procedures.
- Studies have found that students sometimes overestimate how much they understand.
  - Teachers have more experience and a deeper understanding of programming concepts. Students may not realize their shortcomings, especially when their programs are working.
- Failure to imagine the execution process of programming.
- Writing code out of order and assuming the computer remembers a rule previously defined
- Writing code incorrectly - grammar & format
- Leaving out or overlooking specific information that the computer will need
- Conceptual errors - not knowing the correct mathematical formulas or setting them up incorrectly
- Not considering all possible solutions - just because it worked for one scenario does not mean it will work every time
- Taking time to understand the problem and then identifying a solution to the problem
- Jumping into writing your program without a plan
- Using operations from other languages that either don’t work or do an operation the programmer didn’t expect

Solutions

- Once you have a solution, being able to translate that solution to the computer
- Knowing how to test the program for errors
- When your program errors out, knowing how/where to start debugging
- Watching videos of people programming similar things, or visual aids such as flowcharts to help explain what is happening. Visualization of key concepts may help to understand processes.