ITEC 1150 Week 5 Chapter 2.2

LOOPS & RANGE

Chapter 2.1 Recap

REMINDERS AND PROBLEM AREAS FROM LAST WEEK

Nested Blocks

```
1.  score = input('Enter quiz score: ')
2.  if score.isnumeric() is False:
3.      print("Only whole numbers are accepted. Please try again.")
4.  else:
5.      if score >= 90:
6.          name = input('Enter student name: ')
7.          print(f'Great job, {name}!’)
8.          print('Grade: A')
9.              if score >= 99:
10.                 print('You\'ve received one of the best scores ever!')
11.     elif score >= 80:
12.         print('Grade: B')
13.     elif score >= 70:
14.         print('Grade: C')
```

- Each of the indented sections is a **block**.
- A block is related to the line above it, which ends with a colon `:`
- Code inside of a block only runs if the statement above it is **`True`**
- **Blocks can contain other blocks!**
- Note the `else` on line 4 – lines 5-14 are a block inside of the `else`!
- The `if` and `elif` blocks on lines 5, 9, 11, and 13 are called ***nested blocks*** because they are inside another block.
- Note that on line 9 is a nested block inside a nested block! There's no practical limit to how deep nesting can go (although that can get hard to manage; we'll look at how to solve that soon).

Other notes from last week

- Remember that the last item in conditional block is usually `else`. The `else` block will handle any condition not handled by an `if` or `elif`.
- With each program, we may need to add or refine our input validation strategies, to handle common issues. Some of the validation we've wanted to enforce is:
  - An empty value
  - A whole number
  - A positive number
  - A number > another number

Loops

LOOPS ARE THE MOST COMMON CONTROL STRUCTURE, AFTER CONDITIONAL STATEMENTS (IF, ELIF, ELSE)

Why Loops?

- One motivation for computer programming is to do repetitive tasks reliably. Computers, unlike humans, don't make mistakes or get bored (although programmers still do).
- The programs discussed in Chapter 1 were simple lists of instructions.
- In the first part of Chapter 2, we introduced conditional statements (`if`, `elif`, `else`), which allowed our code to 'branch', meaning it could do different things depending on the input.
- This week, we add Looping structures.
- Loops are needed when we need to do repetitive tasks
  - When we want to calculate the wages for all employees
  - Or figure out the profit from all our coffee shop drinks
  - Or keep asking for input until we get a valid entry
  - Or apply the same formatting codes to 100 rows in a table

While loops

Code along! Create an exercise-2.2.py file to try this out during lecture.

- Remember the `apollo.py` program?
- What if we want the user to keep guessing until they get the correct answer?

```
1. answer = input('When did Apollo 11 land on the moon? ')
2. while answer != '1969':
3.     print('Wrong, try again')
4.     answer = input('Enter the year when Apollo 11 landed on the moon: ')
5. print('You are correct!')
```

- Like conditionals, loops have a colon `:`
  - Followed by an indented block
  - Everything indented under the loop will run until the loop condition is `False`.
  - The loop condition must be `True` for the indented loop code to run.

The break statement

- The break statement (line 6) gives us an alternate way to end a loop. This is useful when we have more than one reason for the loop to end.

```
1.  answer = input('When did Apollo 11 land on the moon? ')
2.  while answer != '1969':
3.      print('Wrong. Please try again or enter "q" to quit')
4.      answer = input('Enter the year Apollo 11 landed on the moon: ')
5.      if answer == 'q':
6.          break
7.  if answer == '1969':
8.      print('You are correct!')
9.  else:
10.     print('Sorry you gave up.')
```

- Note that we had to add another `if` / `else` outside our loop to check for the correct answer.
- Unlike the version of this program on the previous slide, the user can exit the loop without having found the answer.

For Loops and the Range Function

- Try this code in a new file called `loop-counter.py` (adapted later for lab).
- It prints a string 5 times, or just a list of numbers

```
1. for count in range(5):
2.     print(f'Hello {count}')
3.
4. for number in range(10):
5.     print(number)
```

- `range(5)` creates a list of numbers from 0 to 4. `count` is a variable; each time through the loop, it will have the next value of the list of numbers.
- Try running this code. What do you notice about the numbering and how is it different than you expect?

Nested Loops

- In your exercise file, add the following code & run it. Try to explain each line by adding a comment to the code. Beware of the indentation!
- What's the difference between the running subtotal & the grand total?

```
1.  while True:
2.      print('Welcome to the coffee shop.')
3.      total = 0.0
4.      items = int(input('How many items is the customer buying? '))
5.      for item in range(items):
6.          price = float(input(f'Enter price in whole dollars for item #{item + 1}: '))
7.          total = total + price
8.          print(f'Running subtotal = ${total :<.2f}')
9.      print(f'Grand total = ${total :<.2f}')
10.     close = input('Time to close? Enter y or n:')
11.     if close == 'y':
12.         break
13. print('Thanks for using the program')
```

- What if we don't know how many times a loop will run?
- For example, we don’t know how many customers we’ll have, or how many items they'll buy.

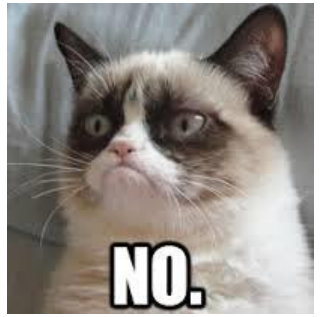
Nesting – Indentation Levels Are Key

- To build on previous programs, let's consider nested loops and conditionals:
  - Often needed – nested loops can deal with the fact that we have a *variable* number of customers, and each one wants a *variable* number of items.
  - You can use loops inside conditionals , conditionals inside loops, and conditionals inside conditionals
  - Indenting correctly is vital, so Python can tell which line of code to run
- You can probably think of ways to improve previous programs, using nesting.
  - In lab we will revisit the `apollo.py` program – what if you want to validate the user entry?
  - What if you want to add validation to the old `coffee_sales.py` code?
  - Later, you'll see a more advanced nested program to prepare for labs.

Another For Loop

- Add this code to your exercise file & run:

```
1. questions = int(input('How many questions do you want to ask Grumpy Cat? '))
2. print('This program thinks that Grumpy Cat will answer as follows: ')
3. for question in range(questions):
4.     print(f'Answer to question #{question + 1} is NO.')
```



- You can use whatever you want as the variable name. This program uses **question** while the previous programs used **count** and **number**. Use whatever makes sense.
- Putting `question +1` in the printout avoids showing `"question 0"`.
  - How else can we accomplish this?

for or while? That is the Question!

- while loops are best when there's a logical condition:

```
while continue == 'y':
while answer != '1969':
```

- for loops are best when you have a list or count:

```
for count in range(10):
for item in ['coffee', 'tea', 'cappuccino']:
```

- You can almost always re-write a for loop as a while loop, and vice versa
- Do you favor one or the other?
- Use the one that makes most sense to you

More About `range()`

- As we've seen, the `range()` function can be used to count.
- And `for` loops can use the output of the range function to control the number of times a loop repeats

- But what if you don't want to start counting at 0?
- Or, what if you want to count by 2s, or by 10s?
- Or, what if you want to count backwards?

More Ways to Use range()

- Perhaps we want our loop to start counting at 50 and count to 99, AND to add each number to the next.
- You can use `range()` with two arguments, where the first is the start and the second is the end.

```
1. print('I can print and add up all the numbers between 50 and 100.')
2. total = 0
3. for number in range(50, 100):
4.     print(number)
5.     total += number # longhand version → total = total + number
6. print(f'The grand total of all the numbers is: {total:,d}.')
```

More Ways to Use range() (cont.)

- Perhaps we only want our loop to display even numbers, or to count by threes.
- We can add a third argument called "step":

```
1. for even_number in range(0, 10, 2):
2.     print(even_number)
3.
   for count_by_three in range(33, 66, 3):
4.     print(count_by_three)
```

Inclusive and Exclusive – a Note on `range()`

- Inclusive means *included*
- Exclusive means *excluded*
- Range is always `range(inclusive, exclusive)`, meaning the first number is included, but the last number isn't:

```
for number in range(1, 5):
    print(number)
```

- That will print 1, 2, 3, 4 but will *not* print 5.
- If you wanted 1-5 printed, you would need to write

```
for number in range(1, 6):
    print(number)
```

- When necessary, the labs will specify inclusive or exclusive ranges—pay attention to this!

Range() Notes Continued

- Off by one issues – the cause of many bugs!
  - As you have seen, you can use a variable like `book + 1` in your print strings, to avoid showing a message about book 0.
  - It's a similar situation with `range()`. By default, the function counts to one number below the stop value, so you often need to add 1 to the stop value.
- Shortcuts
  - Do you prefer to type.... `total = total + price`
  - Or do you like the shortcut... `total += price`

Works with other math operations too!

Looping over other things

- You can loop through the characters in a string too!
- Try this in your exercise file:

```
my_college = 'Minneapolis'
for letter in my_college:
    print(letter)
print('Acronym = ' + my_college[0] + 'C')
```

What's the [0] doing?

Another way to say it… Strings are "iterable." Each character has a place index starting at 0.

M	i	n	n	e	a	p	o	l	i	s
0	1	2	3	4	5	6	7	8	9	10

Validation with Loops

- In the last lab, we had very simple validation: if the input was wrong, we printed a message and the program ended.
- With loops, we can prompt the user to keep trying until they enter valid data:

```
1. value = input("Please enter a whole number larger than 5: ")
2. while value.isnumeric() is False or int(value) <= 5:
3.     value = input("Please enter a whole number larger than 5: ")
```

Putting Together Formats & Loops

- Here is a multiplication table loop, which includes a formatted print command inside the loop (indented code).
- The printing and formatting are repeated each time the loop runs.
- Save in your exercise file & run:

```
print('Here is the 12 times table:')
for number in range(13):
    print(f'\t{number:>2d} times 12 is {number *12:>3d}')
```

```
Here is the 12 times table:
     0 times 12 is   0
     1 times 12 is  12
     2 times 12 is  24
     3 times 12 is  36
     4 times 12 is  48
     5 times 12 is  60
     6 times 12 is  72
     7 times 12 is  84
     8 times 12 is  96
     9 times 12 is 108
    10 times 12 is 120
    11 times 12 is 132
    12 times 12 is 144
```

Nesting Once Again (Lab Help)

- Download the sample file demo2-2_coffee_sales_nested.py and run in PyCharm
- Is this program more complicated or simpler than our previous coffee sales program?
- Let's go through the code and add comments
- For more help on nesting, there are many YouTube videos available. Search "Python nested loops".

```
How many different types of drinks did you sell today? 2

Name of the drink type #1: Coffee
        Number of cups of Coffee size 1 sold: 23
        Price of Coffee size 1: $ 3.99
        Coffee      Sz. 1       23       $3.99       $    91.77
        Number of cups of Coffee size 2 sold: 34
        Price of Coffee size 2: $ 4.99
        Coffee      Sz. 2       34       $4.99       $   169.66
        Number of cups of Coffee size 3 sold: 56
        Price of Coffee size 3: $ 5.99
        Coffee      Sz. 3       56       $5.99       $   335.44

Name of the drink type #2: Tea
        Number of cups of Tea size 1 sold: 17
        Price of Tea size 1: $ 1.99
        Tea         Sz. 1       17       $1.99       $    33.83
        Number of cups of Tea size 2 sold: 21
        Price of Tea size 2: $ 2.99
        Tea         Sz. 2       21       $2.99       $    62.79
        Number of cups of Tea size 3 sold: 34
        Price of Tea size 3: $ 3.99
        Tea         Sz. 3       34       $3.99       $   135.66

Today's Drink Sales for the Reporting Period
Types         Cups Sold                 Total
2               185                   $ 829.15
```