Lecture 2

Python Structure and Basic Data Structures

Solution 1 to Assignment 2

```
1  brand = input("What is your favorite car brand? ")
2  car = input("What is your favorite car from that brand? ")
3  print(car + " is my favorite car from " + brand)
4  print(car + " is my favorite car from " + brand)
5  print(car + " is my favorite car from " + brand)
6  print(car + " is my favorite car from " + brand)
7  print(car + " is my favorite car from " + brand)
```

This would be the most obvious way of doing this assignment based on what we have learned so far. Notice the spacing with the strings this is how it should it be done so you don’t get awkward spacing.

Solution 2 to Assignment 2

```
1  brand = input("What is your favorite car brand? ")
2  car = input("What is your favorite car from that brand? ")
3  print((car + " is my favorite car from " + brand + "\n") * 5)
4
```

This is how I solved this assignment. I noticed that some of you tried to do the same thing but weren't able to get the string to be printed on separate lines.

To solve this issue we need to include the "\n" which means new line at the end of each line

Spacing Issues with Assignment 2

```
1  brand = input("What is your favorite car brand? ")
2  car = input("What is your favorite car from that brand? ")
3  print(car + " is my favorite car from " + brand)
```

- If you had weird spacing when you ran your program it is because you did not format your strings exactly like how I did here.
- Strings and coding have very strict rules and logic that we have to follow to get the results that we want.
- I included spaces after the “?” in the inputs so that when it asks the user to enter something there is already a space between their input and the text.
- I included spaces in front of “is” and after “from” so that there is a space between the user inputs when it prints. Otherwise there will be no space between the car/brand and the string

Something to keep in mind

Although this is something to keep in mind don’t get too hung up on it. For now as long as your code works, it works.

We will be adding more to this list as we move through the course

- Everything that we do has a cost to it. Even the language itself has a cost to using it over another language.
- Just because you CAN do something doesn’t mean you SHOULD
- Reduce redundancy, increase elegance

Some examples of this seen in last lecture

- Using a lot of if statements can work but require a lot more tinkering and ruin efficiency

What is Style?

Just like writing, everyone has their own style and this is because there are many ways to do/say the same thing.

A part of learning how to program will come down to developing your own personal style.

Following up from the previous slide, you can have your own personal style but that doesn’t mean your style should ignore good/ideal practices but instead incorporate them.

You will become very familiar with my personal coding style within the next four weeks but that doesn’t necessarily mean if you didn’t do it like how I did it then you are wrong. If your implementation works the same way then that is perfectly okay.

Boolean Logic (ICS 6B)

- Used in conjunction with conditional statements
- and, or, not
- Understanding boolean logic is important in being able to control the structure and complexity if your conditionals

And

- True and True = True
- False and True = False
- False and False = False

Or

- True or True = True
- True or False = True
- False or False = False

Not

- not True = False
- not False = True

Boolean Logic Cont.

Like other things that we have gone over in last lecture there are many ways to do things but some ways are simply better.

Remember we always want to reduce redundancy and improve readability!

If you compare lines 4/5 and lines 8/10 they do the same thing but lines 4 and 10 are not as redundant and more readable!

Saying “if this is true” is better than saying “if this is true equals to true, true”

```
1  trueValue = True
2  falseValue = False
3
4  if trueValue == True:
5      print("A")
6  if trueValue:
7      print("B")
8  if falseValue == False:
9      print("C")
10 if not falseValue:
11     print("D")
```

Output:

```
A
B
C
D
```

Try it yourself: Boolean Logic

General format of a conditional

if True:

Do task

What do you think this program will output?

```
1  x = 10
2  y = 20
3  z = 30
4
5  if x > 20 and y < 30:
6      print("X is greater than 20 and y is less than 30")
7  if x > 20 or y < 30:
8      print("X is greater than 20 or y is less than 30")
9  if (x > 20 and y < 30) or z == 30:
10     print("This is too long to type")
```

Output:

Try it yourself: Boolean Logic

General format of a conditional

if True:

Do task

What do you think this program will output?

```
1  x = 10
2  y = 20
3  z = 30
4
5  if x > 20 and y < 30:
6      print("X is greater than 20 and y is less than 30")
7  if x > 20 or y < 30:
8      print("X is greater than 20 or y is less than 30")
9  if (x > 20 and y < 30) or z == 30:
10     print("This is too long to type")
```

Output:

```
X is greater than 20 or y is less than 30
This is too long to type
```

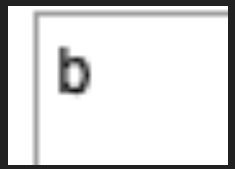
This can get complicated

Boolean Logic can start to get pretty complicated so a personal rule of thumb that I like to follow is to not use more than one logical statement (or, and) per conditional statement.

If you were to use more than one logical statement (or, and) per conditional statement you should make it good practice to implement parentheses to help with readability.

If you don't include parentheses then the "and" statements will take priority over the "or" statements

```
1  x = True
2  y = False
3  z = False
4
5  if x and y or z:
6      print("a")
7  if x or y and z:
8      print("b")
```



As you can see in line 7 if you were to read it from left to right, True or False and False you would get False.

The conditional statement on line 7 is actually True and we know this because it printed "b" and that's because and takes priority over or so it was read as True or (False and False) which is True

Try it yourself: Boolean Logic

```
1  x = True
2  y = False
3  z = True
4
5  if x or y and z:
6      print("A")
7  if x and y or y and z:
8      print("B")
9  if z and (y or x) and z:
10     print("C")
11 if y or z and (x or y) and z:
12     print("D")
13 else:
14     print("None of the above")
```

Solution:

Try it yourself: Boolean Logic

```
1  x = True
2  y = False
3  z = True
4
5  if x or y and z:
6      print("A")
7  if x and y or y and z:
8      print("B")
9  if z and (y or x) and z:
10     print("C")
11 if y or z and (x or y) and z:
12     print("D")
13 else:
14     print("None of the above")
```

Solution:
A C D

range() & counting in programming

Range is a function within the standard library that produces a range of numbers for us given an integer. We can use this mainly for loops.

For Example:

range(5) will produce for us values 0, 1, 2, 3, 4

range(3, 6) will produce for us values 3, 4, 5

Notice how the first number is inclusive and the last number is exclusive. This rule applies to all programming. Everything starts at 0 or the first value inclusive and the last value is always exclusive.

For Loops

for loops are used to execute a command repeatedly.

You can control the number of times a command is executed in a loop by specifying the starting and ending values.

In the example on the right, the temporary variable “i” iterates through the list and takes on each value of the list in order, one at a time.

So, when the loop starts, i=0 and executes the command within the loop. Then i takes on the next value of the list (1) and executes the command within the loop and so on and so forth until the list ends.

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

Output:

For Loops

for loops are used to execute a command repeatedly.

You can control the number of times a command is executed in a loop by specifying the starting and ending values.

In the example on the right, the temporary variable “i” iterates through the list and takes on each value of the list in order, one at a time.

So, when the loop starts, i=0 and executes the command within the loop. Then i takes on the next value of the list (1) and executes the command within the loop and so on and so forth until the list ends.

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

Output:

0

1

2

3

4

Try it yourself: Use a for loop for Assignment 2

Using a for loop for Assignment 2

```
1  brand = input("What is your favorite car brand? ")
2  car = input("What is your favorite car from that brand? ")
3  for i in range(5):
4      print(car + " is my favorite car from " + brand)
```

We can use a for loop for this to print the statement multiple times

While Loop

while loops also execute commands repeatedly but unlike **for loops,** you do not specify a beginning and an end.

In **while loops**, you specify a test condition that determines when the loop stops. Until this test condition is not met, the loop continues to execute.

while loops are especially useful when you don’t know how many times you want the command to repeat.

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

Output:

While Loop

while loops also execute commands repeatedly but unlike **for loops,** you do not specify a beginning and an end.

In **while loops**, you specify a test condition that determines when the loop stops. Until this test condition is not met, the loop continues to execute.

while loops are especially useful when you don’t know how many times you need the command to repeat.

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

Output:

0

1

2

3

4

5

Try it yourself: Use a while loop for Assignment 2

Using a while loop for Assignment 2

```
1  brand = input("What is your favorite car brand? ")
2  car = input("What is your favorite car from that brand? ")
3  i = 0
4  while i < 5:
5      print(car + " is my favorite car from " + brand)
6      i += 1
```

Since a while loop uses a conditional to determine when to stop we use a counter to determine how many iterations the while loop should do.

Introduction to Scope

If you are coming from another language you are most likely use to this general structure

```
if (conditional 1) {
        Do task
        if (conditional 2) {
                Do task
        }
}
```

This is how you would do the same thing in python

```
if conditional 1:
        Do task
        if conditional 2:
                Do task
```

What is Scope?

In simple terms scope is how code is organized in what each statement can and can't access

Anything defined in the global scope (no indentation) can be accessed from anywhere in the program

Anything defined within a statement will remain in that statement's scope and cannot be accessed anywhere else

"The **Global** scope"

Parent statement 1:

Task

Child statement 1:

Task

Child statement 2:

Task

Grandchild statement 1:

Task

Parent statement 2:

Task

Alternative to keyword: and

Sometimes having a lot of Boolean conditionals can make your code confusing to read.
If things start getting cluttered with too many Boolean expressions you can turn them into nested if statements instead

```
1  brian = "cool"
2  trung = "cool"
3
4  if brian == "cool" and trung == "cool":
5      print("cool beans")
6
```

```
1  brian = "cool"
2  trung = "cool"
3
4  if brian == "cool":
5      if trung == "cool":
6          print("cool beans")
7
```

A Quick Note/Reminder

The end of the week is approaching and payments for the course will start being collected Friday - Sunday.

For payment it will be through either venmo/paypal/zelle and more information on this will be announced in the discord later today.