Junior Knights Python I - Week 7

Nested Loops

Agenda

Poll

Debugging

Random Numbers

Nested Loops

Poll: Which Topic Would You Like More Practice With?

If/Else

While Loops

For Loops

Python Operators

Conditionals

If/Else

What is the bug in each of the examples below?

```
x = 5
if x = 5:
        print("x is 5")
```

```
if True
        print("Hello")
```

Nested loops=>

While Loops

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

```
x = 10
while x < 5:
        print(x)
        x += 1
```

Nested loops=>

For Loops

```
for i in range(1, 10):
        print(i)
```

```
n = 10
multiples = []
 for i in range(n):
        if i % 3 == 0:
                multiples.append(i)
print(multiples)  # Expected output: [3, 6, 9]
```

Nested loops=>

Python Operators

The following code should **extract the last digit** of a number, but it's incorrect. Fix it.

```
n = 1234
last_digit = n // 10

print(last_digit)  # Expected output: 4
```

The following code is supposed to return the **number of full hours** in a given number of minutes, but it's producing incorrect results. Fix it.

```
minutes = 130
hours = minutes / 60

print(hours)  # Expected output: 2
```

Nested loops=>

Conditionals

This code is supposed to print all numbers that are divisible by 3 and 5, but it is printing more numbers than the expected. What is the issue?

```
n = 30
for i in range(1, n):
        if i % 3 == 0 or i % 5 == 0:
                print(i)
```

Nested loops=>

Random Numbers

- To use the random module, we must import it and seed it:

```
import random
random.seed()
```

`random.randrange(start, stop, step)`

- Returns a random number between the given range, can skip numbers using step, stop is *not* inclusive

`random.randint(start, stop)`

- Returns a random number between the given range, stop *is* inclusive

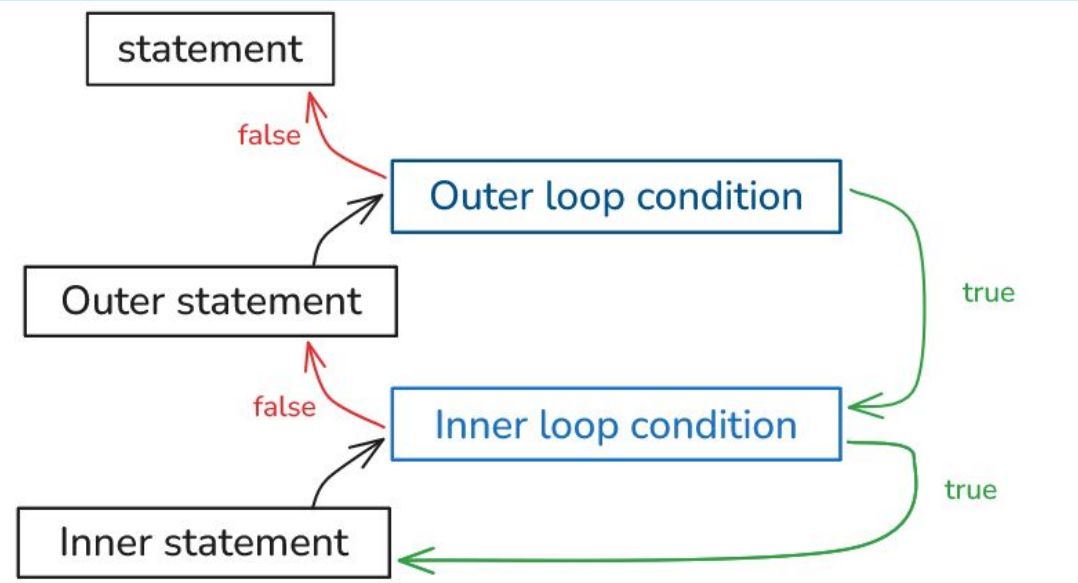
Nested Loops

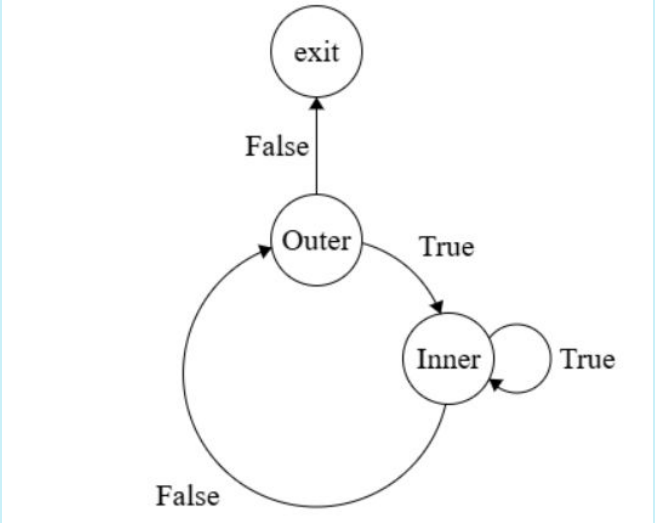
- In any for/while loop, you can include loops within these loops
- Code reads top down; the first loop is entered first, runs as normal
  - Whenever a second loop is encountered, that loop iterates until completion
  - Once that loop is finished, the rest of the first loop statements are executed… and then it loops again!
    - After this, the process repeats, so the second loop will begin again, iterate until completion

- Your second/inner loop will run to completion (loop several times) once per every iteration on the first loop. So if your inner loop runs 3 times, and your outer loop runs 5, the statements within the inner loop will execute **15 times!**

Nested Loops

- Diagrams are very helpful when imagining such loops.





Nested Loops

- Such loops can be helpful when dealing with tables or matrices– if you need to perform operations in a 2d space, you normally will be creating a nested loop.

- Some examples:
  - Multiplication tables
  - Tic-Tac-Toe games
  - Clocks
  - Coordinates
  - … and so much more!

Syntax:

Outer_loop Expression:

Inner_loop Expression:

Statement inside inner_loop

Statement inside Outer_loop

Nested Loops

- Here, we introduce a unique kind of for loop syntax. In addition to the **for x in range(a/a,b/a,b,c)**, when iterating through an array, you can also utilize the **for x in y**, where x is the name for the variable that loops (can be any valid variable name), and y is the array you would like to loop through. Where the range function is not inclusive at the endpoint, this will span **all elements of the array**.

- What will be the output of this code?

```
a = [1, 2]
b = [4, 5]

for i in a:
  for j in b:
      print(i, j)
```

Live Coding

And/Or Expressions

Boolean values

- A Boolean is a type of variable
- It can be one of two things: “True” or “False”
- Boolean logic is the foundation of what and/or statements operate under

- Booleans can be declared like any other variable:

```
x = 9
y = True
```

And/Or Expressions

- Or
  - An or statement evaluates to “True” if either or both of its conditions are true
  - Only evaluates to “False” when no conditions are met

- And
  - An and statement only evaluates to “True” if both conditions are true
  - Evaluates to “False” if one or both the statements are false

- Not
  - If the statement the not encloses evaluates to “True”, the Not operator (!) changes it to “False”, and vice versa

- The **order of precedence** from **highest to lowest** is **Not, And, and then Ors**. If you have a compound and/or statement, it is good practice to **always use parentheses** to group statements properly.

And/Or Expressions

Easy

```
a = True
b = False
print(a and b)

x = False
y = True
print(x or y)

q = False
r = True
print(not q or r)
```

Medium

```
x = True
y = False
z = True
print((x and y) or (y or z))

a = False
b = False
c = True
 print(not (a and b) or (b and c))
```

Tricky

```
x = True
y = False
z = True
print(not ((x and y) or (not z and y)))
```

And/Or Expressions

The importance of parentheses:

```
a = True
b = False
c = True

print(a and b or c)
```

```
a = True
b = False
c = True
print(a and (b or c))
```