COMP1516

LESSON 3: BRANCHING, WHILE LOOPS, CLI

Agenda – Lesson 3

- Quick Review
  - Last Week's Topics Lab
  - Pre-Reading
- Quiz 2
  - Review Answers
- Branching (i.e., Conditionals)
- While Loop
- CLI
- Lab 3 – In Class
- Homework – Lab 3 Take Home and Pre-Reading

Quiz 2

- This is an individual closed book assessment, please do your own work
- You have 25 minutes to complete it
- We will go over the answers afterwards

Course Outline

#	Topics	Quiz	Lab	Assignment
1	Python Setup, Variables, Expressions, Naming Conventions		Lab 1	
2	Logical Operators, Functions and Modules, Main Function	Quiz 1	Lab 2	
3	Branching and Loops	Quiz 2	Lab 3	
4	Strings	Quiz 3	Lab 4	
5	Lists, Tuples, Dictionaries, and Sets	Quiz 4	Lab 5	
6	Pandas	Midterm	Lab 6	Assignment 1 Due

Lesson #3 Learning Outcomes

- If statements/(Conational Structure) (if, if else, and if elif else)
- Boolean Operators
- Boolean expression
- Indefinite Iterations (while loop)
- Command Line Interface (CLI)

if Statement

- Sequential execution of the statements: default mode of execution where statements are executed *one after another* in the order they appear in the code.

```
print("First statement")
print(“Second statement")
print(“Third statement")
```

- Sometimes the behaviour of program needs to change depending on a certain condition.
- What are conditional statements?
  - Conditional statements allow a program to execute certain code only if specific conditions are met.
- **if statements** are used to check conditions and change the behaviour of the program accordingly; they can be used to skip code sections, or to select one section out of many:

```
statement1
if something is True:
    statement2    # this statement may or may not be executed
statement3
```

if statement

- General syntax:

  if *boolean expression*:

  statement

  statement

  etc.

- The statements will execute ONLY IF the result of the Boolean Expression is True.

- **Note:** In Python, indentation is a crucial aspect of the language's syntax. Unlike many other programming languages that use braces or keywords to define blocks of code, Python uses indentation to define the structure and flow of the program.

if statement

- Example:

```
if age_in_years >= 16 :
    print("you can drive")
```

if/else Statement

- A second if statement form exists for two possibilities:
  - Do one path if the statement is True;
  - Do the other path if the statement is False.

General Syntax:

if *Boolean expression*:

    statements to do because the “if” is True

else:

    statements to do because the “if” is False

if/else Statement

Example:

```
if number%2 == 0:
    print("the number is even")
else:
    print("the number is odd")
```

if/else Statement

```
first_name = input("What's your name? ")
age_in_years = int(input("How old are you"))

if age_in_years >= 16:
    print("you are old enough to drive\nbe safe\nhave fun\ndon't over speed")
else:
    print(f'Hello {first_name}')
    print("you are not old enough to drive")
    print("you have to wait", 16 - age_in_years, "more years", "!!!")
```

if/else Statement

- When you write an if-else statement, follow these guidelines for indentation:
  - Make sure the if block and the else block are aligned (same level of indentation).
  - The if block and the else block are each followed by a colon and then an intended block of statements.
  - Make sure the statements inside each block are consistently indented.

Chained conditionals (if / elif / else)

- Sometimes there are more than two possibilities, so we need more than two branches.
- Example:

```
if number > 0:
    print("the number is positive")
elif number < 0 :
    print("the number is negative")
else:
    print("the number is 0")
```

Chained conditionals (if / elif / else)

```
first_name = input("What's your name? ")
age_in_years = int(input("How old are you"))

if age_in_years > 16:
    print("you are old enough to drive\nbe safe\nhave fun\ndon't over speed")
elif age_in_years == 16:
    print("You are barely onld enough to drive")
    print("go take some lessons")
else:
    print(f'Hello {first_name}')
    print("you are not old enough to drive")
    print("you have to wait", 16 - age_in_years, "more years", "!!!")
```

if/ elif/ else

- elif is an abbreviation of else if.
- Ultimately *exactly one* branch will be executed, and the rest will be skipped.
- There is no limit to the number of elif branches.
- Practice: Letter Grades
    - A: >=90
    - B: 80~89
    - C: 70~79
    - D: 60~69
    - F <60

Solution:

Any issues?

```
grade = int(input("Final Grade: "))
print(grade)
if grade>=90:
    print("A")
elif grade >=80:
    print("B")
elif grade >=70:
    print("C")
elif grade >=60:
    print("D")
else:
    print("F")
```

Nested if statements

If statements can be nested to implement complex logic

Example:

To validate that a test mark is between 0 and 100 inclusive:

```
if test_mark >= 0:
    if test_mark <= 100:
        print(test_mark," is a valid test mark")
    else:
        print(" test mark cannot be greater than 100")
else:
    print("test mark cannot be negative")
```

But do not indent too many times. That code is too complex to follow.

Practice

Rewrite the Letter Grade

```
grade = int(input("Final Grade: "))
if grade >100 | grade<0:
    if grade>=90:
        print("A")
    elif grade >=80:
        print("B")
    elif grade >=70:
        print("C")
    elif grade >=60:
        print("D")
    else:
        print("F")
else:
    print("Invalide")
```

Distinct if Statements

- Each if statement whether it is an if or if/else or if/elif/else statement is considered one decision making statement.
- The first branch that returns true will be executed and the other branches will be ignored.
- Multiple distinct if statements are multiple decisions because each if statement will be evaluated individually and the branch that returns true of every if statement will be executed.

Distinct if statements - example

Distinct if statement

```
age = 18

if age < 16:
    print("youth")

if age >= 16:
    print("Old enough to drive")

if age >= 18:
    print("Old enough to vote")
```

All three if's are evaluated always

Output:

```
Old enough to drive
Old enough to vote

Process finished with exit code 0
```

If elif statement

```
age = 18

if age >= 18:
    print("Old enough to vote")
elif age >= 16:
    print("Old enough to drive")
else:
    print("Youth")
```

Runs until one if or elif is True or until it reaches the else

Output:

```
Old enough to vote

Process finished with exit code 0
```

Practice:

```
a = 33
b = 32

# First conditional block
if b > a:
   print("b is greater than a")
# Otherwise if a equals b

elif a == b:
   print("a and b are equal")
# This condition is entered if all previous blocks are not selected.

else:
   print("* Program output *")
   print("b is less than a")

# Second conditional block
if not a == b:

   print("b and a are not equal")
```

The output?

```
a = 33
b = 34

# First conditional block
if b > a:
   print("b is greater than a")
# Otherwise if a equals b

elif a == b:
   print("a and b are equal")
# This condition is entered if all previous blocks are not selected.

else:
   print("* Program output *")
   print("b is less than a")

# Second conditional block
if not a == b:

   print("b and a are not equal")
```

The output?

Short-Circuit Evaluation

- Python evaluates if statement Boolean expression from left to right
- Sometimes evaluating the left part is sufficient to determine the end result therefore the right part will not be evaluated.
- This is called Short-Circuit Evaluation.

Short-Circuit Evaluation: and

- Example:

```
number = 5
value = 7
if (number > 10 and value > 0) :
    # Do something
```

- The first(left) part of the if statement expression results in False therefore, the second part will not be evaluated because
- False and anything ➔False

Short-Circuit Evaluation: or

- Similarly,

```
number = 5

value = 7

if (number < 10 or value < 0):

        # Do something
```

- The first(left) part of the if statement expression results in True therefore, the second part will not be evaluated because True or anything ➔ True

Short-Circuit Evaluation

Running the following code :

```
if  1/0 :
    print("Division by 0 attempt")

print(" if statement was executed")
```

Results in :

```
    if 1/0 :
ZeroDivisionError: division by zero

Process finished with exit code 1
```

Short-Circuit Evaluation

Running the following code :

```
number = 5
if number> 10 and 1/0 :
    print("Division by 0 attempt")

print(" if statement was executed")
```

Results in:

```
 if statement was executed

Process finished with exit code 0
```

Short-Circuit Evaluation

- We can use the concept of short-circuit evaluation as a “guard”:

- if num != 0 and (total / num) >= 0.0:
- 	# Do something

- The num != 0 check prevents the total / num from being evaluated when it would result in a divide by zero error.

Introduction to Loops

- What are loops?
  - Loops are constructs that allow you to repeat a block of code multiple times.
  - Useful for iterating over sequences, performing repetitive tasks, and automating processes.
- Types of Loops in Python
  - while loop: repeats as long as a condition is True.
  - For loop: iterates over a sequence

if statement vs. while statement

if conditional statement:

    do this

    do that

    do this too

while conditional statement:

    do this

    do that

    do this too

A while loop is a self-repeating if statement

if statement vs. while statement

- A block of code inside an “if statement” gets executed zero times, or one time.

- A block of code inside a “while statement” gets executed zero times, one time, … 500 times, … infinity times. It depends on the logic.

Repetition Statements - while loop

- Many algorithms require executing a set of statements repetitively; these statements are called loops.
- Python provides two types of loops: Condition-controlled and Count-Controlled.
- Condition-controlled loop is called a “while loop”.
- A “while-loop” allows you to repeat a block of code until a condition is no longer True.

Repetition Statements - while loop

- The condition is given before the loop body.
- The condition is checked **before each execution** of the loop body; it is like a self-repeating if statement.
- Typically, the while loop is used when it is impossible to determine the exact number of loop iterations in advance.
- General Syntax:

```
while <condition>:
    block of code
```

Repetition – while loops

- Example: to display the numbers from 1 to 10:

```
number = 1
while number <= 10 :
    print(number, end=" ")
    number = number + 1
```

- Output:

```
1 2 3 4 5 6 7 8 9 10
Process finished with exit code 0
```

Repetition Statements: while loop

```
# print the even numbers from 1 to 100

index = 1

while index <= 100:
    if index % 2 == 0:
        print(index)
    index += 1  # or index = index + 1
```

Repetition Statements: while loop

- This is how the while loop is executed:

```
number = 1
while number <= 10:
    print(number)
    number =  number + 1
```

the condition will be checked,

Repetition Statements: while loop

- This is how the while loop is executed:

```
number = 1
while number <= 10:
    print(number)
    number = number + 1
```

the condition will be checked

if True the while block(body) runs

Repetition Statements- while loop

- This is how the while loop is executed:

```
number = 1
while number <= 10:
    print(number)
    number = number + 1
```

After the last statement of the while block is executed, the computer goes back to the condition and checks it again.

If True the while block(body) runs, if it is false the while loop stops.

Infinite Loop

- An infinite loop occurs when a program keeps executing within one loop, never leaving it.
- This happens if the condition will never be False.

- Example:

```
number = 1
while number <= 10:
    print(number)
```

Stopping a while Loop

- The while loop condition should eventually evaluate to False.

- Use **break** statement to terminate the loop, the flow of control will jump to the statement directly after the while loop.

- Use **exit(0)** to terminate the program(rarely used).
- Use **return** to exit from a function.
- Use **continue** to skip an iteration of the while loop.

Stopping a while Loop

- Example of using break statement:

```
def main():
    count = 0
    while count < 10:
        count += 1
        if count == 5:
            break
        print("current count value is",count)

    print("done looping ")

if __name__ == "__main__" :
    main()
    print("main method has finished execution")
```

Output:

```
current count value is 1
current count value is 2
current count value is 3
current count value is 4
done looping
main method has finished execution
```

Stopping a while Loop

- Example of using return statement

```
def main():
    count = 0
    while count < 10:
        count += 1
        if count == 5:
            return
        print("current count value is",count)

    print("done looping ")

if __name__ == "__main__" :
    main()
    print("main method has finished execution")
```

Output:

```
current count value is 1
current count value is 2
current count value is 3
current count value is 4
main method has finished execution
```

Stopping a while loop

- Example of using continue statement:

```
def main():
    count = 0
    while count < 10:
        count += 1
        if count == 5:
            continue
        print("current count value is",count)

    print("done looping ")

if __name__ == "__main__" :
    main()
    print("main method has finished execution")
```

Output:

```
current count value is 1
current count value is 2
current count value is 3
current count value is 4
current count value is 6
current count value is 7
current count value is 8
current count value is 9
current count value is 10
done looping
main method has finished execution
```

range() function

- Function range() creates a ‘range’ type object that represents a sequence of integers with a start, end and step.
- General syntax:

  range(start, stop, step)

- Update to three arguments(start, end, step)
- Combinations:
  - **range(end)** # start is 0 and step is 1; e.g. range(50) gives a range from 0 to 49, by 1s
  - **range(start, end)** # step is 1; e.g. range(30, 40) gives a range from 30 to 39, by 1s
  - **range(start, end, step)** # e.g. range(45, 57, 2) gives a range 45, 47, 49, 51, 53, 55
- ***Note***: Range() does not work with float values, because float cannot be interpreted as an integer.

range() function - Examples

Range	Generated Sequence	Explanation
range(5)	0 1 2 3 4	Every integer from 0 to 4.
range(0, 5)	0 1 2 3 4	Every integer from 0 to 4.
range(3, 7)	3 4 5 6	Every integer from 3 to 6.
range(10, 13)	10 11 12	Every integer from 10 to 12.
range(0, 5, 1)	0 1 2 3 4	Every 1 integer from 0 to 4.
range(0, 5, 2)	0 2 4	Every 2nd integer from 0 to 4.
range(5, 0, -1)	5 4 3 2 1	Every 1 integer from 5 down to 1
range(5, 0, -2)	5 3 1	Every 2nd integer from 5 down to 1

Output: range(2, -6, 2) ?

print(list(range(2, 6, 2)))

range() function with while loop

- Using a range() function in a while loop

```
count =1

while count in range (1, 11):
    print("Current count value is ", count)
    count+=1
```

Output:

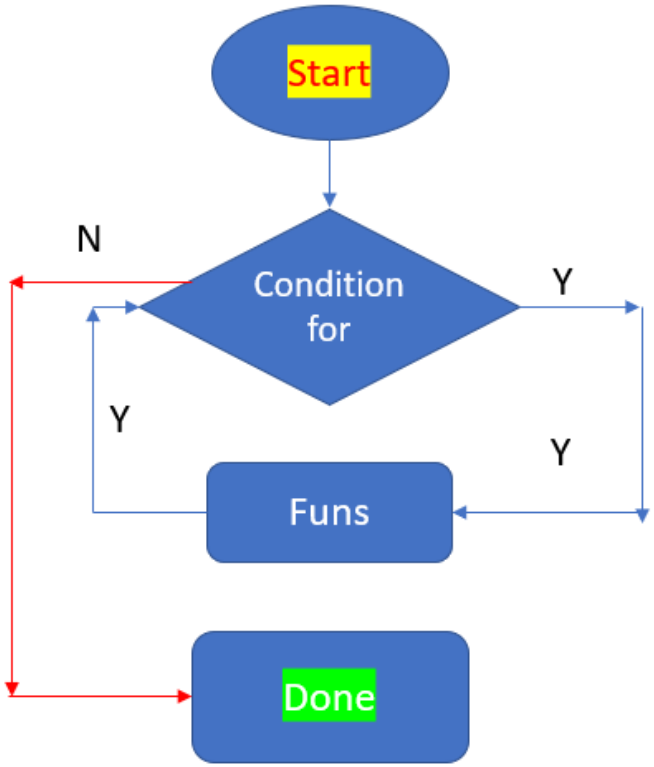
```
Current count value is  1
Current count value is  2
Current count value is  3
Current count value is  4
Current count value is  5
Current count value is  6
Current count value is  7
Current count value is  8
Current count value is  9
Current count value is  10


** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Definite Iteration – for loop

- A for loop allows us to iterate over a series of instructions for a specific number of times. For loops are useful whenever a counter is needed.
- For loops can be used to iterate over a sequence/container (string, list, tuple) at a time, assigning the next element to the variable that can be used in the loop body.
- For loop syntax:

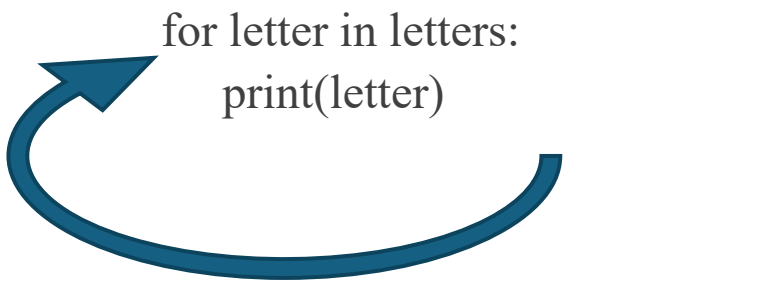
```
for <var> in <iterable> :
    <statements>
```



For Loop

- letters = [“a”, “b”, “c”, “d”]

a
b
c
d



letter

For Loop

- Using a for loop with a string:

```
def main():
    #for loop examples

    my_string = "banana"
    for ch in my_string:
        print(ch)
```

Output:

```
b
a
n
a
n
a
```

For Loop

- Using for loop with a list:

```
def main():
    #for loop examples

    my_list = ["apple","banana","kiwi","cherry","orange"]
    for item in my_list:
        print(item)
```

Output

```
apple
banana
kiwi
cherry
orange
```

For Loop

Using for loop with a tuple:

```
marks = (74,89,68,95,70)

def AverageGrade(tup):
    sum=0
    for grade in tup:
        print("The marks is ", grade)
        sum+=grade
    return round(sum/len(tup), 2)
print("Average Grade is ", AverageGrade(marks))
```

Output:

```
The marks is  74
The marks is  89
The marks is  68
The marks is  95
The marks is  70
Average Grade is  79.2
```

For Loop

- Using for loop with a range
- Example:

```
for number in range(1,11):
    print(number,end=" ")
```

Output:

```
1 2 3 4 5 6 7 8 9 10
```

while vs for loops

```
index = 0
while index < 100:
    # Loop body statements
    index += 2
```

```
for index in range(0, 100, 2):
    # Loop body statements
```

General Rules:

- Use a **for loop** when you can compute the number of iterations before entering the loop
- Use a **for loop** when access elements from a sequence container
- Use a **while loop** when the number of iterations is not computable before entering the loop

while vs for loops

Demo:

while loop use case

- Calculator that continuously prompts the user to do a calculation

for loop use case

- One time calculation of interest over the duration of a term deposit

Demo

```python
def calculator():
    while True:
        # Prompt the user for an operation
        print("Select operation:")
        print("1. Add")
        print("2. Subtract")
        print("3. Multiply")
        print("4. Divide")
        print("5. Exit")

        choice = input("Enter choice (1/2/3/4/5): ")

        if choice == '5':
            print("Exiting the calculator.")
            break

        # Ensure valid choice
        if choice not in ('1', '2', '3', '4'):
            print("Invalid input. Please try again.")
            continue

        # Get numbers from the user
        num1 = float(input("Enter first number: "))
        num2 = float(input("Enter second number: "))

        # Perform the chosen operation
        if choice == '1':
            print(f"The result is: {num1 + num2}")
        elif choice == '2':
            print(f"The result is: {num1 - num2}")
        elif choice == '3':
            print(f"The result is: {num1 * num2}")
        elif choice == '4':
            if num2 != 0:
                print(f"The result is: {num1 / num2}")
            else:
                print("Error: Division by zero.")
```

break vs. continue Statement

Ex:

```
for x in range(0, 4):
  print(x)

  if(x >= 2):
    break
```

Output?

```
for x in range(0, 4):

  if(x > 1):
    continue
  print(x)
```

Output?

```
for x in range (0,4):
  print("x value is :", x)
  if x>1:
    continue
  print(x)
```

Nested Loop (Loops within Loops)

```
numDays  = 7
numWeeks = 4

for week in range(1, numWeeks + 1):
    # end="" avoids pointer going to new line.
    print("week " + str(week) + ": ", end="")
    # Show days of week
    for day in range(1, numDays + 1):
      print("day" + str(day) + "   ", end="");
    print("") # Goes to new line
```

```
week 1: day1  day2  day3  day4  day5  day6  day7
week 2: day1  day2  day3  day4  day5  day6  day7
week 3: day1  day2  day3  day4  day5  day6  day7
week 4: day1  day2  day3  day4  day5  day6  day7
```

Modify the code in Example 11 so the output becomes the following:

```
week 1: Sunday1   Monday2   Tuesday3   Wednesday4   Thursday5   Friday6   Saturday7
week 2: Sunday1   Monday2   Tuesday3   Wednesday4   Thursday5   Friday6   Saturday7
week 3: Sunday1   Monday2   Tuesday3   Wednesday4   Thursday5   Friday6   Saturday7
week 4: Sunday1   Monday2   Tuesday3   Wednesday4   Thursday5   Friday6   Saturday7
```

Hint: You will want to use *if* and *elif*.

CLI Command Line Interface

- A Python program is run from the command-line (i.e., the Windows or Linux command prompt or terminal)

- In PyCharm, we have three options to run our programs
  - PyCharm Console (preferred)
  - Python Console using runfile () command
  - Terminal using python command

CLI Command Line Interface

- Example of running a program from Python console using runfile()

  Given the following python script

```
#hello.py
def main():
    print("hello world!")

if __name__ == "__main__":
    main()
```

Running the script from Python console:

CLI Command Line Interface

- Example of running a script from Windows Command Prompt using command python

Given the following python script

```python
#hello.py
def main():
    print("hello world!")

if __name__ == "__main__":
    main()
```

Running the script from Windows command Prompt:

```
(venv) C:\Users\Rana\PycharmProjects\DataTypes>python hello.py
hello world!

(venv) C:\Users\Rana\PycharmProjects\DataTypes>
```

CLI Command Line Interface

- For a “production” Python program, it would be run from a Windows Command Prompt or Linux Terminal.
- For our development, we will use the PyCharm console.

You can pass inputs to your program through the Command Line Interface – we will look at this when we do Lists in Week 5

Lab 3 and Next Week Homework

Lab 3 has two parts:

- Part A – To be done in class.
- Part B – To be done at home (or in class if you finish Part A early). Due in the Lab 3 dropbox, along with the code for Part A, Wednesday at midnight

I will start you off on Part A in class you should be able to show the working code on your laptop.

Pre-Reading for Lesson 4:

- Posted to the Learning Hub tomorrow
- As usual, questions can be asked through E-mail.