Iteration III

Recall that a loop allows you to execute the same statements multiple times. Previously we've studied `for` loops. This week we will study `while` loops, which continue to execute as long as a condition is true.

Warm-up

1. Rewrite the program below to have only a single if statement. Your solution must have exactly one `if`, zero or more `elif` branches, and zero or one `else` branch. (In your solution, you need only rewrite the part below the comment. You may draw the control flow chart and use it in the simplification.)

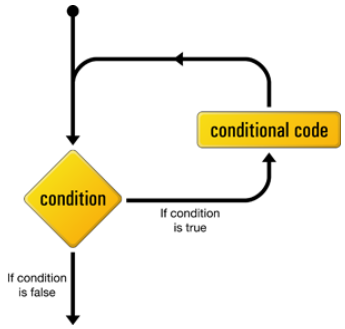
```
def main() -> None:
    response = input("Is it raining out? ")
    rainy = response == 'yes'
    wind = int(input("What is the wind speed? "))
    # rewrite the program from here down...
    if rainy:
        if wind > 30:
            print ("Wear your rain slicker!")
        else:
            print ("Bring your umbrella!")
    else:
        if wind > 30:
            print ("Wear your windbreaker!")
main()
```

```
if rainy and wind > 30:
    print ("Wear your rain slicker!")
elif rainy:
    print ("Bring your umbrella!")
elif wind > 30:
    print ("Wear your windbreaker!")
```

Model 1 `while` Statements

A more general looping structure is the `while` statement. The instructor will trace the following program using the debug interface.

```
def main() -> None:
    i = 0
    while i < 3:
        print("the number is", i)
        i = i + 1
    print("goodbye")
main()
```



2. What must the value of the Boolean expression (after the `while`) be in order for the first print statement to execute? True

3. Circle the statement that changes the variable `i` in the above code. `i = i + 1` is circled

4. What happens to the value of the variable `i` during the execution of the loop?

It increments by one each time the loop body is executed.

5. Explain why the loop body does not execute again after it outputs "the number is 2".

The variable `i` then becomes 3, which causes the condition `i < 3` to be false.

6. Suppose the order of the statements in the loop body is reversed:

```
while i < 3:
    i = i + 1
    print("the number is", i)
```

a) How does the order impact the output displayed by the `print` function?

It prints the numbers 1,2,3 instead of 0,1,2.

b) Does the order impact the total number of lines that are output?

No it does not—either way, there are 3 lines.

7. Identify three different ways to modify the code so that the loop only executes twice.

You can change the first line to `i = 1`. Or you can change the condition to `i < 2`. Or you can change the last line to `i = i + 2`.

8. Describe the three parts of a `while` loop that control the number of times the loop executes.

The first part initializes the variable or condition.
The second part tests whether the end has been reached.
The third part updates the variable or condition.

9. What would happen if you comment out the statement `i = i + 1`?

It prints "the number is 0" forever - infinite loop.

When writing a `while` loop, it's helpful to answer a few questions before you start:

- *What needs to be initialized before the loop?*
- *What condition must be true for the loop to repeat?*
- *What will change so that the loop eventually ends?*

10. Consider the function `add(n)` that prompts the user for n numbers and returns the sum of these values. For example, when `add(5)` is called, the user is asked to input five numbers. If the user inputs 3, 1, 5, 2, and 4, the function would return the value 15.

a) Describe the variable that needs to be initialized before the loop begins. i = 0

b) Describe the Boolean expression that must be true for the loop to continue. i < n

c) Describe what will need to change so that the loop will eventually end. i = i + 1

d) Now list what else needs to happen inside the body of the loop for you to calculate the sum of the user input.

1) Prompt the user to input a number, and 2) add that number of a running total.

e) Given your previous answer, are there any other values that need to be initialized before the start of the loop?

Yes, the running total should be initialized to zero.

f) Write the code for `add(n)` below:

```
1  def add(n: int) -> int:
2      sum = 0
3      i = 0
4      while i < n:
5          num = int(input("Enter number: "))
6          sum += num
7          i += 1
8      return sum
```

Model 2 Sentinel-value Pattern with `while` loops

The previous problem employs the sentinel-value pattern. The sentinel value is a value used to signal the end of the loop (in this case, the sentinel value was `n`).

11. Rewrite the following code using a `while` loop and employing the sentinel-value pattern:

```
def sum_list(lst -> list) -> int:
    sum = 0
    for x in lst:
        sum = sum + x
    return sum
```

```
def sum_list(lst: list) -> int:
    sum = 0
    i = 0
    while i < len(lst):
        sum = sum + lst[i]
        i += 1
    return sum
```

Model 3 Tracing with `while` loops and Infinite loops

12. What does this function print with input 3? If there is an infinite loop, indicate the first 4 lines of what the program prints, and state that there is an infinite loop.

```
def stop_at_one(x: int) -> int:
    count = 0
    while x != 1:
        print(x)
        if x % 2 == 1:
            x = 3 * x + 1
        else:
            x = x // 2
```

```
3
10
5
16
8
4
2
```

13. What does this function print with input 3? If there is an infinite loop, indicate the first 4 lines of what the program prints, and state that there is an infinite loop.

```
def stop_at_one(x: int) -> int:
    count = 0
    while x != 1:
        print(x)
        if x % 2 == 1:
            x = 2 * x + 1
        else:
            x = x // 2
```

```
3
7
15
31
Infinite loop
```

Model 4 Validating-input Pattern with `while` loops

You can also use a while loop when you want to validate input; when you want to make sure the user has entered valid input for a prompt. Let's say you want a function that asks a yes-or-no question:

```
def get_yes_or_no(message: str) -> None:
    valid_input = False
    while not valid_input:
        answer = input(message)
        answer = answer.upper() # convert to upper case
        if answer == 'Y' and answer == 'N':
            print(f'Valid input! Thanks for answering {answer}')
            valid_input = True
        else:
            print('Invalid input! Please enter Y for yes or N for no.')

def main() -> None:
    get_yes_or_no('Do you like lima beans? Y)es or N)o: ')
main()
```

14. Trace the code assuming that the user enters `'yes'`, `'no'`, `'y'` and `'n'` in this order?

```
Invalid input! Please enter Y for yes or N for no.
Invalid input! Please enter Y for yes or N for no.
Valid input! Thanks for answering Y
Valid input! Thanks for answering N
```

15. Will the program terminate? If your answer is no, then provide the fix.

The code does not terminate because it's an infinite loop. To fix it change `and` to `or` in Line 6.

Model 5 `while` loop writing code practice

16. Write a function, called `sublist`, that takes in a list of numbers as the parameter. In the function, use a while loop to return a sublist of the input list. The sublist must contain the same values of the original list up until it reaches the number 5 (it must not contain the number 5).

```
1 def sublist(lst: list) -> list:
2     sub_lst = []
3     i = 0
4     while i < len(lst) and lst[i] != 5:
5         sub_lst += [lst[i]]
6         i += 1
7     return sub_lst
```

17. Write a function, called `sublist`, that takes in a list of strings as the parameter. In the function, use a while loop to return a sublist of the input list. The sublist should contain the same values of the original list up until it reaches the string "STOP" (it should not contain the string "STOP").

```
1 def sublist(lst: list) -> list:
2     sub_lst = []
3     i = 0
4     while i < len(lst) and lst[i] != 'STOP':
5         sub_lst += [lst[i]]
6         i += 1
7     return sub_lst
```

18. Write a function called `beginning` that takes a list as input and contains a while loop that only stops once the element of the list is the string 'bye'. What is returned is a list that contains up to the first 10 strings, regardless of where the loop stops. (i.e., if it stops on the 32nd element, the first 10 are returned. If "bye" is the 5th element, the first 4 are returned.) ***If you want to make this even more of a challenge, do this without slicing.***

```
1 def beginning(lst: list) -> list:
2     sub_lst = []
3     i = 0
4     while i < len(lst) and i < 10 and lst[i] != 'bye':
5         sub_lst += [lst[i]]
6         i += 1
7     return sub_lst
```