Fundamentals of Programming 1

Lecture 6

Aurelia Power, ITB, 2018

Iteration/Repetition

- Many tasks can be accomplished by repeating certain steps over and over.
- For instance, a brick layer continuously lays block upon block until the wall been constructed is complete.
- Similarly, some programming problems work by repeatedly carrying out the same actions.
- In fact, the power of computers comes from ability to do repetitive tasks and programmers can use this to solve many problems – so… iteration is very important

For Example

- We are so tired after a long day's work that we want to write the message

  "***Programming is hard ***"

- 10 times or 100 times or 1000 …

Iteration/Repetition

One Solution is to use the sequence structure…

```
print (“Programming is hard work“)
print (“Programming is hard work“)
print (“Programming is hard work“)
print (“Programming is hard work“)
print (“Programming is hard work“)
print (“Programming is hard work“)
print (“Programming is hard work“)
print (“Programming is hard work“)
print (“Programming is hard work“)
print (“Programming is hard work“)
print (“Programming is hard work“)
## … and so on until you have 100 statements
```

But it still doesn’t solve the issue of boredom … and if we simply copy and paste that statement 100 or 1000 times we run into another problem: the program becomes unreadable with so many lines of code…

Better Solution!

We can use the Iteration Control Structure

- **Iteration** structure allows a section of code to be repeated over and over again.
- The programming structure that is used to control this repetition is often called a **loop**.
- Iteration is achieved by using **iterative control statements** to repeatedly execute a set of instructions.
- There are two **types of loops/iteration statements** that we will cover:

  - **while** statement/loop;
  - **for** statement/loop.

The 'while' Loop - Syntax

- The **while loop/statement** is an iterative structure that repeatedly executes a set of statements based on whether a certain Boolean condition is satisfied: as long as that expression resolves to true (the condition remains true) the loop will repeatedly execute.

```
while boolean_condition:
    instruction(s) to be repeated go here
```

Header of the loop

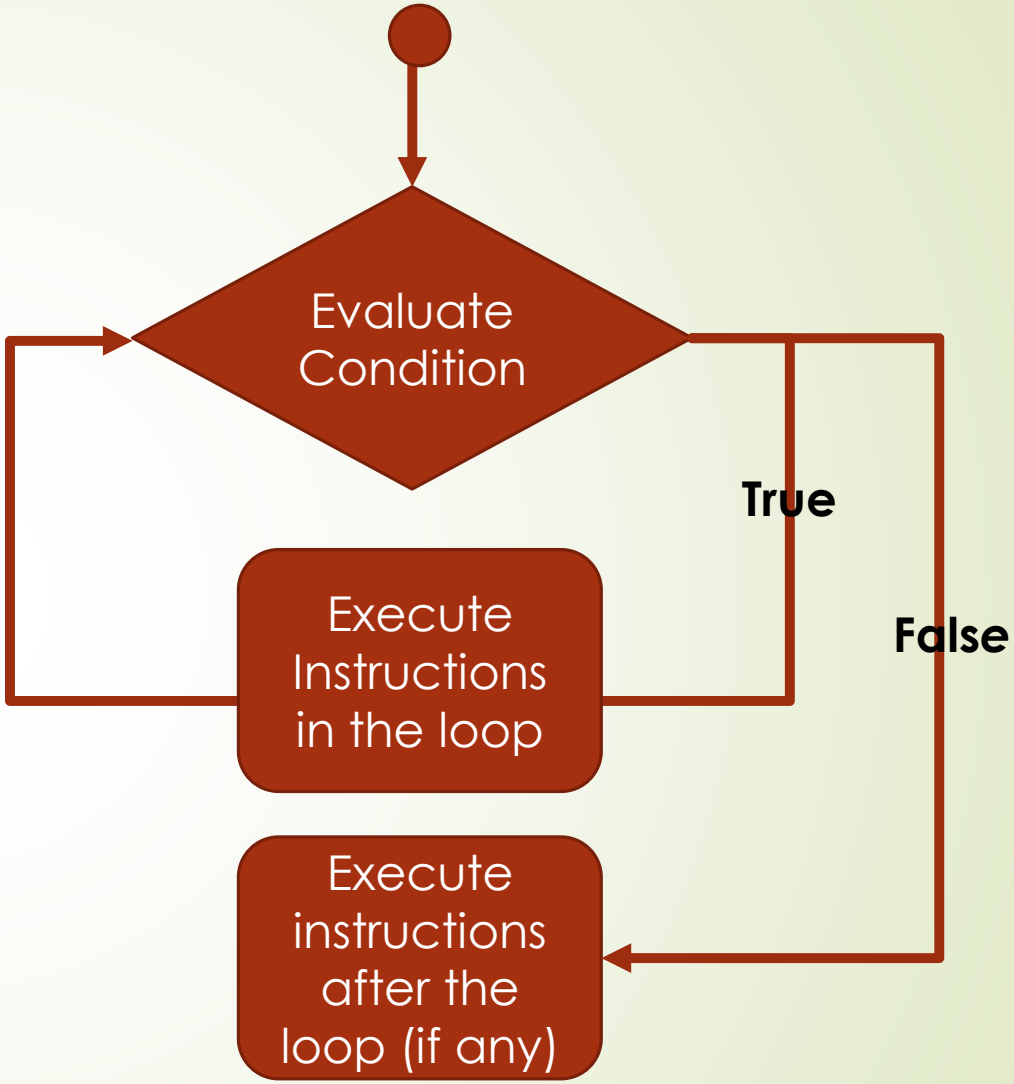
Body of the loop

- The header contains the following elements in this exact order:
  1. the keyword **while**
  2. followed by the **Boolean condition**
  3. followed by the colon (:)

- The body contains the **instructions** that need to be repeated
  - These instructions **must be indented**.

The ‘while’ Loop

- The Boolean condition is repeatedly checked to see whether is satisfied:
  - If the condition is satisfied (the Boolean expression resolves to True), the statements in the loop body will be executed and then the condition is evaluated again
  - If the condition is not satisfied (the Boolean expression resolves to False), the statements in the loop body will no longer be executed
- So the loop stops when the Boolean condition becomes False



The ‘while’ Loop – displaying ‘programming is hard’ 100 times

- We need to repeat the print statement 100 times and to do so we need to:
  - Evaluate the Boolean condition repeatedly and ensure that it holds True 100 times
  - Make sure that the loop stops after 100 iterations by making that condition become False
  - In code, we then need a variable that keeps track of the number of repetitions and this variable has to be repeatedly updated in each iteration until it makes the condition False

PSEUDOCODE:

1. Create a variable/counter to keep track of the number of iterations and give it initially the value of 1
2. REPEAT WHILE/AS LONG AS the counter is less than or equal to 100
   a) output ‘programming is hard’
   b) Update the value of the counter by increasing by 1

The ‘while’ Loop – displaying ‘programming is hard’ 100 times

File Edit Format Run Options Window Help

```
## create a variable for keeping track of repetitions
counter = 1
## repeat as long as the value of counter is less or equal to 100
while counter <= 100:
    ## print the message
    print('Programming is hard')
    ## update the value of counter
    counter = counter + 1
```

- It will output 100 lines of ‘Programming is hard’
- How many times is the body of the loop executed?
  100 times
- How many times is the condition evaluated?
  101 times
- Can we change the initial value of the counter to 0?
  Yes, but then we also need to change the Boolean expression to counter < 100 to ensure exactly 100 repetitions.
- Can we start with a counter of 7?
  Again, yes, but the Boolean expression must be changed to counter < 107

Displaying ‘programming is hard’ 100 times

- **Before entering the loop: counter = 1**
- **Let’s track the value of counter every time the condition is evaluated:**

Evaluation no.	Boolean: counter <= 100	Iteration	Display ‘programming…’	counter = counter + 1
1	1 <= 100 (True)	1	Programming is hard	counter = 1 + 1 (2)
2	2 <= 100 (True)	2	Programming is hard	counter = 2 + 1 (3)
3	3 <= 100 (True)	3	Programming is hard	counter = 3 + 1 (4)
…	…	…	…	…
100	100 <= 100 (True)	100	Programming is hard	counter = 100 + 1 (101)
101	101 <= 100 (False)	-	-	-

- **After the loop is finished: counter is 101**

The ‘while’ Loop – letting the user decide how many times to display ‘programming is hard’

- This time the user decides how many times to print ‘programming is hard’
- We need an extra variable to hold that

PSEUDOCODE:

1. Prompt, take user input, and convert it to an integer value.
2. Create a variable/counter to keep track of the number of iterations and give it initially the value of 1
3. REPEAT WHILE/AS LONG AS the counter is less than or equal to user input
   a) output ‘programming is hard’
   b) Update the value of the counter by increasing by 1

```
## Prompt, take user input, and convert it to an integer value.
n = int(input("How many times do you want to display 'programming is hard'? "))
## create a variable for keeping track of repetitions
counter = 1
## repeat as long as the value of counter is less or equal to n
while counter <= n:
    ## print 'programming is hard'
    print("programming is hard")
    ## update the value of counter
    counter += 1
```

The ‘while’ Loop – displaying the numbers 1 to 10

- We need to repeat the print statement 10 times and to do so we need to:
  - Evaluate the Boolean condition repeatedly and ensure that it holds True 10 times
  - Make sure that the loop stops after 10 iterations by making that condition become False
  - In code, we then need a variable that keeps track of the number of repetitions and this variable has to be repeatedly updated in each iteration until it makes the condition False
  - We need to output the value of the counter in each repetition

PSEUDOCODE:

1. Create a variable/counter to keep track of the number of iterations and give it initially the value of 1
2. REPEAT WHILE/AS LONG AS the counter is less than or equal to 10
   a) output the value of the counter
   b) Update the value of the counter by increasing by 1

Displaying the numbers 1 - 10

File Edit Format Run Options Window Help

```
## create a variable for keeping track of repetitions
counter = 1
## repeat as long as the value of counter is less or equal to 10
while counter <= 10:
    ## print the value of the counter
    print(counter, end=' ')
    ## update the value of counter
    counter = counter + 1
```

```
===== RESTART: C:/Use
1 2 3 4 5 6 7 8 9 10
>>>
```

- NOTE: in the print function **you can specify the end of each line**
  - the default (when you don't specify any) is the new line
  - Here we specified to end with a space: end=' '
  - What will it output if we specified: end='**'
- How many times is the body of the loop executed?
  10 times
- How many times is the condition evaluated?
  11 times
- EXERCISE: display the numbers 16 to 25

Displaying the numbers 1 - 10

- **Before entering the loop: counter = 1**
- **Let's track the value of counter every time the condition is evaluated:**

Evaluation no.	Boolean: counter <= 10	Iteration	print(counter, ' ')	counter = counter + 1
1	1 <= 10 (True)	1	1	counter = 1 + 1 (2)
2	2 <= 10 (True)	2	2	counter = 2 + 1 (3)
3	3 <= 10 (True)	3	3	counter = 3 + 1 (4)
…	…	…	…	…
10	10 <= 10 (True)	10	10	counter = 10 + 1 (11)
11	11 <= 10 (False)	-	-	-

- **After the loop is finished: counter = 11**
- **But that value is not printed because the condition is False so the print statement is not executed then**

The ‘while’ Loop – displaying the numbers 5 – 0 backwards

- We need to repeat the print statement 6 times and to do so we need to:
  - Evaluate the Boolean condition repeatedly and ensure that it holds True 6 times
  - Make sure that the loop stops after 6 iterations by making that condition become False
  - In code, we then need a variable that keeps track of the number of repetitions and this variable has to be repeatedly updated in each iteration until it makes the condition False
  - We need to output the value of the counter in each repetition

PSEUDOCODE:

1. Create a variable/counter to keep track of the number of iterations and give it initially the value of 5
2. REPEAT WHILE/AS LONG AS the counter is greater than or equal to 5
   a) output the value of the counter
   b) Update the value of the counter by decreasing by 1

Displaying the numbers 5 - 0

```
## create a variable for keeping track of repetitions
counter = 5
## repeat as long as the value of counter is greater or equal than 0
while counter >= 0:
    ## print the value of the counter
    print(counter, end=' ')
    ## update the value of counter
    counter = counter - 1 ## can re-write it as counter -= 1
```

```
===== RESTART:
5 4 3 2 1 0
>>>
```

- How many times is the body of the loop executed?
  6 times
- How many times is the condition evaluated?
  7 times
- EXERCISE: display the following numbers in this exact order: 23 21 19 17 15 13 11 9 7

Displaying the numbers 5 - 0

- **Before entering the loop: counter = 5**
- **Let's track the value of counter every time the condition is evaluated:**

Evaluation no.	Boolean: counter >= 5	Iteration	print(counter, ' ')	counter = counter - 1
1	5 >= 0 (True)	1	5	counter = 5 - 1 (4)
2	4 >= 0 (True)	2	4	counter = 4 - 1 (3)
3	3 >= 0 (True)	3	3	counter = 3 - 1 (2)
4	2 >= 0 (True)	4	2	counter = 2 - 1 (1)
5	1 >= 0 (True)	5	1	counter = 1 - 1 (0)
6	0 >= 0 (True)	6	0	counter = 0 - 1 (-1)
7	-1 >= 0 (False)			

- **After the loop is finished: counter = -1**
- **But that value is not printed because the condition is False so the print statement is not executed then**

The ‘while’ loop – summing the numbers 1 - 5

EXAMPLE: calculating and displaying the sum of the numbers 1 to 5 which is 15.

PSEUDOCODE:

1. Create a variable to hold the sum (or total) and assign the value of 0 initially.

2. Create another variable counter to keep track of the number of repetitions and assign the value of 1 initially.

3. REPEAT WHILE/AS LONG AS the counter is less than or equal to 5
   a) Update the value of total by adding the value of the counter to it
   b) Update the value of the counter by 1

4. Output the value of the sum.

While loop: Summing the numbers 1 to 5 - code

File Edit Format Run Options Window Help

```
## create a variable for sum or total
total = 0
## create a variable for keeping track of repetitions
counter = 1
## repeat as long as the value of counter is less or equal to 5
while counter <= 5:
    ## update the value of total
    total = total + counter
    ## update the value of counter
    counter = counter + 1
## output the value of total
print('The sum of the numbers 1 - 5 is:',  total)
```

```
>>>
===== RESTART: C:/Users/Aurelia Power
The sum of the numbers 1 - 5 is: 15
>>>
```

The ‘while’ loop – summing the numbers 1 - 5

- **Before entering the loop: total = 0; counter = 1**
- **Let’s track the values of each variable every time the condition is evaluated:**

Evaluation no.	Boolean: counter <= 5	Iteration	total = total + counter	counter = counter + 1
1	1 <= 5 (True)	1	total = 0 +1 (1)	counter = 1 + 1 (2)
2	2 <= 5 (True)	2	total = 1 +2 (3)	counter = 2 + 1 (3)
3	3 <= 5 (True)	3	total = 3 +3 (6)	counter = 3 + 1 (4)
4	4 <= 5 (True)	4	total = 6 +4 (10)	counter = 4 + 1 (5)
5	5 <= 5 (True)	5	total = 10 +5 (15)	counter = 5 + 1 (6)
6	6 <= 5 (False)	-	-	-

- **After the loop: total = 15; counter = 6**
- **So when we print the value of total, 15 is going to be outputted**

The WHILE loop – Input Validation

- The while loop is well suited for **user-input validation**
- Let's go back to our age program!!
- So far, we are able to enter any sort of numbers, including negative values, but age cannot be negative.
- In this case, we want to prompt the user to re-enter the value until the age is positive!!!
- So, as long as the input is negative (that is, smaller than 0), the program should do the following:
    1. Re-display the prompt to enter age
    2. And take input again and assign to the variable age
- Then the program can continue its execution as planned.

The WHILE loop – Input Validation

File Edit Format Run Options Window Help

```
## prompt and take input for age and put it into the variable age, then co
age = int(input ("Enter your age:  "))

## as long as the age is negative, prompt, take input and convert
while age < 0:
    age = int(input ("Re-enter your age:  "))

## output age
print('you are', age, 'years old')

## if the age is greater than 18 tell the use that can get a driving licer
if (age >= 18):
    print('you can get a driving license...')
## otherwise tell the user is underage
else:
    print('you are underage...')
## finally output a closing message
print("See you later !!");
```

The WHILE loop – Input Validation

Sample output:

```
Enter your age:  -12
Re-enter your age:  -7
Re-enter your age:  -100
Re-enter your age:  12
you are 12 years old
you are underage...
See you later !!
```

Evaluation no.	Boolean: age < 0	Iteration	age = int(input ("Re-enter your age: "))
1	-12 < 0 (True)	1	Re-enter your age: (age -> decided by next input which is in this case -7)
2	-7 < 0 (True)	2	Re-enter your age: (age -> decided by next input which is in this case -100)
3	-100 < 0 (True)	3	Re-enter your age: (age -> decided by next input which is in this case 12)
4	12 < 0 (False)	-	-

- **How many iterations?**
  3
- **How many evaluations?**
  4

The WHILE loop – Input Validation

Sample output:

```
>>>
===== RESTART: C:/Users/Aurelia P
Enter your age:  -12
Re-enter your age:  -7
Re-enter your age:  -100
Re-enter your age:  12
you are 12 years old
you are underage...
See you later !!
>>>
```

Some Questions to think about…

- What would happen if the condition of the while was age > 0??

The program would not allow user to input positive integers, and re-display the prompt until the user enters a negative int !!!

- How can you further improve your program to ensure that the user enters ages within a reasonable age, say between 0 and 300 (we don't expect anyone to live beyond 300…)?

We change the Boolean expression of the while to age < 0 or age > 300

Definite vs. Indefinite Loops

- A **definite loop** is a loop when the number of repetitions is known before the loop is executed
- Examples include: displaying the numbers 1-10, summing up the numbers 1- 10, displaying 'programming is hard' n times
- In all these cases, before the loop is executed, the number of repetitions is known, whether as a fixed literal value or decided by the user.
- Also known as counter-controlled loops
- An **indefinite loop** is a loop in which the number of the times it will be executed is not known.
- For example, the number of times a user may get the wrong input for age (i.e. negative age) is not known before the loop is entered.
- Also known as sentinel-controlled loops.

Counter-Controlled Repetition another example

- **Problem:** *I want to find out the average of correct questions in the FOP1 MCQ for group1. There were 16 students in group 1 that took the test and the grades represent the number of question the students got right (out of 15).*

- To compute the average I need to add all the grades together first, and then divide that sum by the number of students.

- To implement this program, I need to carry out 3 steps repeatedly:

1. **Input the grade** for each student;
2. **Add each grade** to the overall sum.
3. **Keep track of the total** of all grades inputted;

After this, I can compute and output the **average mark.**

- Because the number of times I need to repeat the 3 steps is determined by the number of students: 16 , I can use a definite loop/counter-controlled repetition to input the grades one at a time.
- A variable called a counter (or control variable) controls the number of times a set of statements will execute.

Pseudocode

```
Set total to zero
Set grade counter to 1

While grade counter is less than or equal to 16
    Prompt the user to enter the next grade
    Input the next grade
    Add the grade into the total
    Add one to the grade counter

Set the class average to the total divided by 16
Print the class average
```

- A total is a variable used to accumulate the sum of several values. Variables used to store sums (results of additions) are normally initialized to zero before being used in a program.
- A counter is a variable used to keep track of how many grades have been entered so far.
- The average is computed as total / counter
- What datatype would the average be???

  float

Counter controlled repetition – python code

```
## create a variable for sum or total
total = 0
## create a variable for keeping track of repetitions
counter = 1
## repeat as long as the value of counter is less than or equal to 16
while counter <= 16:
    ## prompt, take user input and convert the mark
    mark = int(input("enter mark for student " + str(counter) + ": "))
    ## update the value of total
    total = total + mark ## can be re-written as total += mark
    ## update the value of counter
    counter = counter + 1 ## can be re-written as counter += 1
## output the value of average
print('The average mark is', total/16)
```

Counter controlled repetition – sample output

```
>>>
===== RESTART: C:/Users/Aurelia Po
enter mark for student 1: 10
enter mark for student 2: 12
enter mark for student 3: 7
enter mark for student 4: 2
enter mark for student 5: 5
enter mark for student 6: 8
enter mark for student 7: 15
enter mark for student 8: 12
enter mark for student 9: 3
enter mark for student 10: 13
enter mark for student 11: 5
enter mark for student 12: 4
enter mark for student 13: 9
enter mark for student 14: 10
enter mark for student 15: 12
enter mark for student 16: 3
The average mark is 8.125
>>>
```

- What does this line do??

```
mark = int(input("enter mark for student " + str(counter) + ": "))
```

- Could I have started the counter at 16?

Yes, but the Boolean condition must be re-written, as well as the prompt for the user.

- How can I do that in code??

There are 2 solutions: one incrementing the counter, the other decrementing the counter… because you can go backwards as well!!

Counter controlled repetition and the problem of infinitely executing statements…

- With counter controlled repetition it is important to update the value of the counter inside the body of the loop… otherwise you will end up with an **infinite loop**
- **An infinite loop** is a loop that runs forever and can be stopped only by killing the program or restarting the computer
- If there are output statements in the program, then reams and reams of output flash by on the screen. Otherwise, the program just sits there and hangs, seeming to do nothing.

```
total = 0
counter = 1
while counter <= 16:
    ## prompt, take user input and convert the mark
    mark = int(input("enter mark for student " + str(counter) + ": "))
    ## update the value of total
    total = total + mark
## output the value of average
print('The average mark is', total/16)
```

The counter will remain 1, and the condition will always evaluate to true… so it will run forever; so make sure that you add counter+=1;

Counter controlled repetition and the problem of never executing statements…

- We also need to make sure that we have the correct condition:
- For instance, what would happen to the following code?

```
total = 0
counter = 1
while counter > 16:
    ## prompt, take user input and convert the mark
    mark = int(input("enter mark for student " + str(counter) + ": "))
    ## update the value of total
    total = total + mark
    ## update the value of counter
    counter = counter + 1
## output the value of average
print('The average mark is', total/16)
```

The condition is False… so the loop statements will never execute

- The body of the loop (the indented statements) will never be executed, because the loop will never be entered since the condition is false right from the very beginning, so total will remain 0 and consequently average will be 0.0

Indefinite Loops - Sentinel-Controlled Repetition

Let's generalise the MCQ average problem and consider the following problem:

Develop a class-averaging program that will process an arbitrary number of grades/marks each time the program is run.

- In the previous example, the number of grades(16) was **known** in advance
- However, now there is **no indication** of how many grades are to be entered.
- The program must process an **arbitrary** number of grades.

How can the program determine when to stop inputting grades?

We can use a special value called a sentinel value (aka flag) that will signal the end of inputting data (the marks/grades in this example)

How does the sentinel value work??

- The user types in grades until all grades have been entered.
- The user types in the sentinel value (usually something that cannot be part of the data) to indicate that there are no more grades to enter.
- So the **sentinel value** is a value that signals the end of the data entry, making the Boolean condition of the loop become false.

Pseudocode

```
Set total to 0
Set counter to 0
Prompt user to enter grade or stop to exit,
take user input and store it into variable mark
REPEAT WHILE/AS LONG AS mark != 'stop'
        convert the mark to int
        add mark to the total
        increase the counter by 1
        prompt user to input mark again
Compute and display average
```

- We still need the total to store the grades added together;
- We still need a counter, but this time it will not be used to determine whether the condition is true, that is the number of repetitions, but to compute average;
- Instead, the input itself stored in variable mark will be used to compare it to 'stop', and if it is 'stop', the condition will become false and the body of the loop will no longer be entered and its statements no longer executed;

Program Code

```
## create a variable for sum or total
total = 0
## create a variable for keeping track of the number of grades
counter = 0
## prompt user to enter grade or exit
mark = input("Enter mark or 'stop' to exit: ")
## repeat as long as the value of grade is not stop
while mark != 'stop':
    ## convert the mark
    mark = int(mark)
    ## update the value of total
    total = total + mark ## can be re-written as total += mark
    ## update the value of counter
    counter = counter + 1 ## can be re-written as counter += 1
    ## prompt user again
    mark = input("Enter mark or 'stop' to exit: ")
## output the value of average
print('The average mark is', total/counter)
```

```
Enter mark or 'stop' to exit: 12
Enter mark or 'stop' to exit: 3
Enter mark or 'stop' to exit: 14
Enter mark or 'stop' to exit: 15
Enter mark or 'stop' to exit: 1
Enter mark or 'stop' to exit: 2
Enter mark or 'stop' to exit: 5
Enter mark or 'stop' to exit: stop
The average mark is 7.428571428571429
>>>
```

- Why did I (in the loop body) update the counter and the total before I prompt the user to re-enter grade?
- Because I have already entered mark once (before I enter the loop) so that also needs to be used to calculate average…

Your Turn…

- Assuming that a is 2 and n is 4, what is the output of the following block of code?

```
r = 1;  i = 1;
while  i <= n:
        r = r * a;
        i+=1;
print(r, i, a, n)
```

Use a table to keep track of values.

- Trace the following code. What error do you observe?

```
n = 1
while n != 30:
        print(n)
        n = n + 10
```

Use a table to keep track of values

- What is the output of the following loop?

```
n = 10
while n >= 0:
        n -= 3
        print(n)
```

Use a table to keep track of values.

- Trace the following code. What error do you observe?

```
lower = 0; upper = 3
while lower >= upper:
        print(lower)
        lower += 1
```

Your Turn…

- Assuming that a is 2 and n is 4, what is the output of the following block of code?

```
r = 1; i = 1
while i <= n  | | r <= n:
    r = r * a
    i += 1
print(r, i, n, a)
```

Use a table to keep track of values.

- Trace the following code. What error do you observe?

```
n = 0
while n <= 0 and n >=7:
    print(n)
    n += 1
```

- What is the output of the following loop? What error do you observe?

```
n = 4
while n % 2 != 0:
    n += 1
    print(n)
```

- Trace the following code. What error do you observe?

```
k = 0
while k < 10:
    print(k)
    k *= 2
```