- Week 4: Iteration:
- while loops
  - Common errors
  - Sentinel Values
  - Common Loop Algorithms

Condition controlled loop

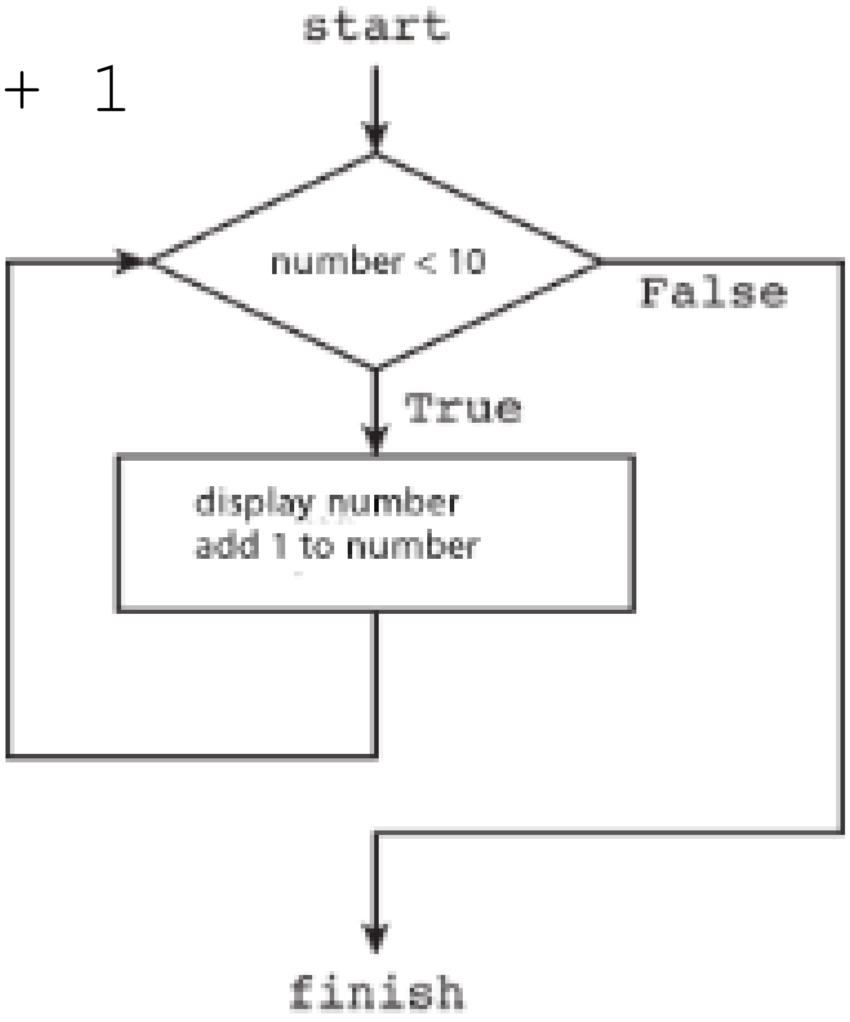
- E.g., print out the value of a number.

This first variable is the **number** variable and you should initialise it before starting your while loop (e.g., give it a value).

If the condition is true (in this case if the variable is less than 10) then execute the next two lines of code.

Update number variable within loop

```
number = 0
while number < 10 :
    print(number)
    number = number + 1
number = 0
while number < 10 :
    print(number)
    number = number + 1
```



Code with Flowchart

Python's use of indentation

- The Python indentation for the while "block" is not cosmetic. A sequence of lines that have the same indentation forms a block. Here, both lines get run or neither of them do. The block is ended by the first line that follows it with no indentation.

```
number = 0
while number < 10 :
    print(number)
    number = number + 1
print('Done!')
```

The indentation indicates a "block" of code.

The first non-indented line marks the end of the block.

Common Error - Infinite Loops

```
counter = 1
while counter <= 10 :          # Executes 5 passes
   print(counter)
   counter = counter + 2
```

```
counter = 1
while counter != 10 :           # Runs forever
   print(counter)
   counter = counter + 2
```

```
counter = 1
while counter != 9 :          # Runs forever
   print(counter)             # counter not updated
counter = counter + 2         # in loop body
```

Common Error – Off-by-One Error

```
counter = 0
while counter < 10 :      # Q1.How many passes?
   <do something>
   counter = counter + 1
```

```
counter = 1
while counter <= 10 :     # Q2.How many passes?
   <do something>
   counter = counter + 1
```

```
counter = 1
while counter < 10 :      # Q3.How many passes?
   <do something>
   counter = counter + 1
```

```
counter = 0
while counter <= 10 :     # Q4. How many passes?
   <do something>
   counter = counter + 1
```

ANSWERS – Off-by-One Error

```
counter = 0
while counter < 10 :       # Executes 10 passes
   <do something>
   counter = counter + 1
```

```
counter = 1
while counter <= 10 :      # Executes 10 passes
   <do something>
   counter = counter + 1
```

```
counter = 1
while counter < 10 :       # Executes 9 passes
   <do something>
   counter = counter + 1
```

```
counter = 0
while counter <= 10 :      # Executes 11 passes
   <do something>
   counter = counter + 1
```

while Loop Review

- Initialize variables before you test
  - The condition is tested BEFORE the loop body
  - Something inside the loop should change a variable used in the test
- Watch out for common errors:
  - Infinite loops, Off-by-One Errors,

  ____

- We will now look at using sentinel values to denote the end of a data set
- Preventing divide by zero
- Common Loop Algorithms

Processing Sentinel Values

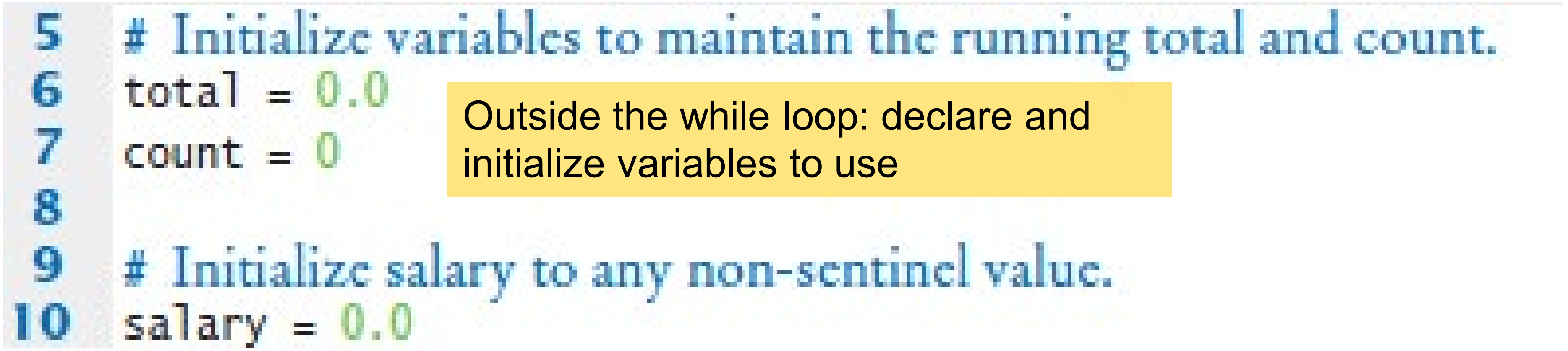
- A **sentinel value** can guarantee termination of a loop when processing sequential data. The **sentinel value** can detect the end of the data set when there is no other means to do so.
- It denotes the end of a data set, but it is not part of the data.
- What value will end this loop?

```
salary = 0.0
while salary >= 0 :
   salary = float(input())
   if salary >= 0 :
       total = total + salary
       count = count + 1
```

Averaging a Set of Values

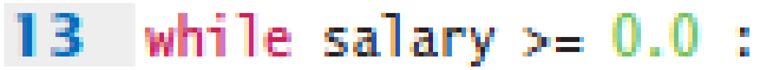
- Declare and initialize a ‘total’ variable to 0.0
- Declare and initialize a ‘count’ variable to 0
- Declare and initialize a ‘salary’ variable to 0.0
- Prompt user with instructions
- Loop until sentinel value is entered
  - Save entered value to input variable (‘salary’)
  - If salary is not -1 or less (sentinel value)
    - Add salary variable to total variable
    - Add 1 to count variable
- Make sure you have at least one entry before you divide!
  - Divide total by count and output.

Sentinel.py (1)



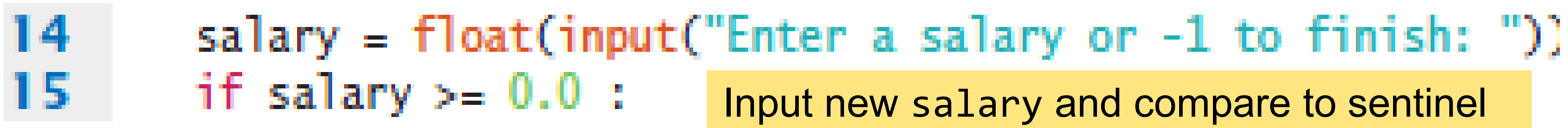
```
# Initialize variables to maintain the running total and count.
total = 0.0
count = 0

# Initialize salary to any non-sentinel value.
salary = 0.0
```

Outside the while loop: declare and initialize variables to use



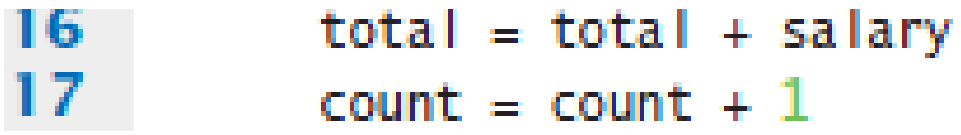
```
while salary >= 0.0 :
```

Since `salary` is initialized to 0, the while loop statements will execute at least once



```
    salary = float(input("Enter a salary or -1 to finish: "))
    if salary >= 0.0 :
```

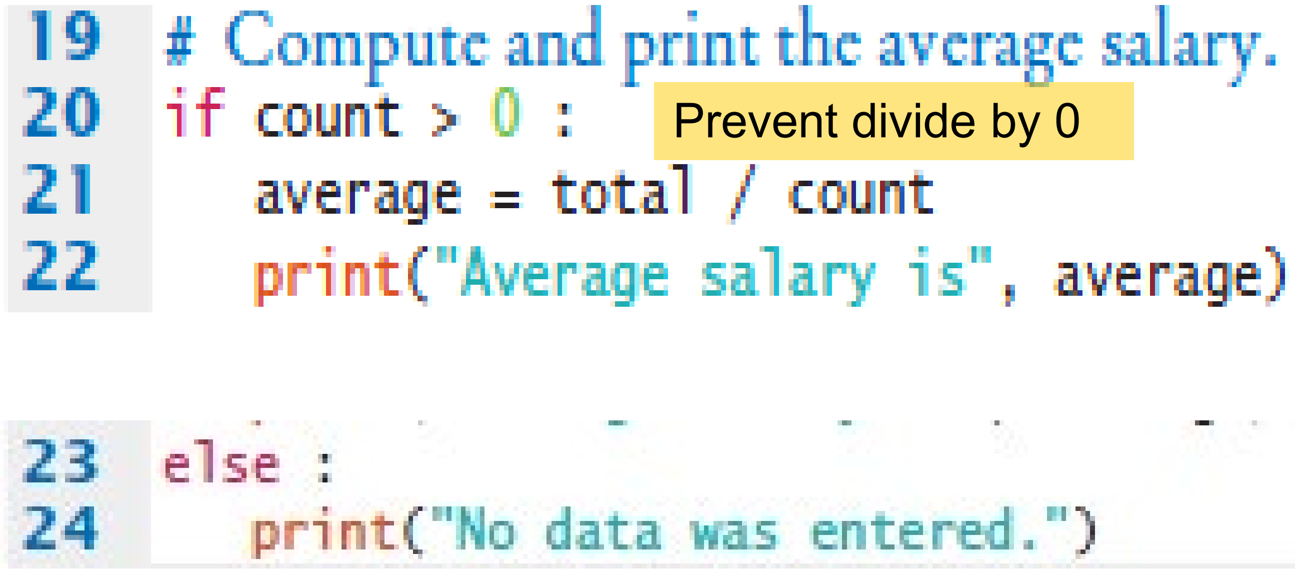
Input new `salary` and compare to sentinel



```
        total = total + salary
        count = count + 1
```

Update running `total` and `count` (to calculate the average later)

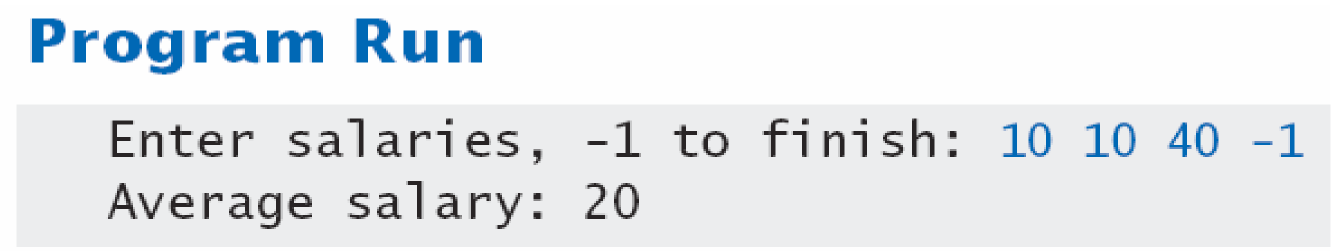
Sentinel.py (2)



```
19  # Compute and print the average salary.
20  if count > 0 :
21      average = total / count
22      print("Average salary is", average)
23  else :
24      print("No data was entered.")
```

Prevent divide by 0

Calculate and output the average salary using the `total` and `count` variables



Program Run

```
Enter salaries, -1 to finish: 10 10 40 -1
Average salary: 20
```

Question - What is the importance of line 15?

Priming Read

- Some programmers don’t like the “trick” of initializing the input variable with a value other than a sentinel.

```
# Set salary to a value to ensure that the
# loop executes at least once.
salary = 0.0
while salary >= 0 :
```

- An alternative - change the variable with a read before the loop.

```
salary = float(input("Enter salary or -1 to exit "))
while salary >= 0 :
```

Modification Read

- The input operation at the bottom of the loop is used to obtain the next input.

```
# Priming read
salary = float(input("Enter salary or -1 to exit "))

while salary >= 0.0 :
    total = total + salary
    count = count + 1
    # Modification read
   salary = float(input("Enter salary or -1 to exit "))
```

Boolean Variable as Flag

- A Boolean variable can be used to control a loop
  - Sometimes called a ‘flag’ variable

```
done = True   # Initialize done to execute loop
while done :
  value = int(input("Enter mark or -1 to exit "))
  if value < 0:
      done = False    # Set done to False if
                      # sentinel value found
  else :
      print('loop again')
```

Common Loop Algorithms 1

- Sum Values: Write a program that contains a while loop that will sum the float values entered by a user until the user enters a number less than zero. Then print the total.
- Here is part of the code:

```
total = 0.0
value = float(input("Enter value: "))
while (1)____________
    (2)_______________
    (3)______________
(4)__________
```

ANSWER - Sum Example

- Exercise – 1. Sum Values: Write a program that contains a while loop that will sum the float values entered by a user until the user enters a number less than zero. Then print the total.

```
total = 0.0
value = float(input("Enter value: "))
while value >= 0 :
    total = total + value
    value = float(input("Enter value: "))
print(total)
```

Common Loop Algorithms 2

Average of Values

- Total the values
- Initialize count to 0
  - Increment per input
- Check for count greater than 0 before divide!

Exercise - Fill in missing code: (1) , (2) and (3)

```
total = 0.0
count = 0
value = float(input("Enter value "))
while value >= 0 :
   total = (1)______________
   count = (2)______________
   value = float(input("Enter value "))

if count > 0 :
   (3)______________
else :
   average = 0.0

print(average)
```

ANSWER - Average Example

Average of Values

- Total the values
- Initialize count to 0
    - Increment per input
- Check for count equal to 0 before divide!

```
total = 0.0
count = 0
value = float(input("Enter value "))
while value >= 0:
   total = total + value     #(1)
   count = count + 1         #(2)
   value = float(input("Enter value "))

if count > 0 :
   average = total / count    #(3)
else :
   average = 0.0

print(average)
```

Dry run a program to find the problem

There is a problem with this program. Track down where the problem is by tracing your program's execution.

Trace this program to find where the problem is and then re-write the program so that it works as intended

```
health = 10
trolls = 0
damage = 3

while health != 0:
    trolls += 1
    health -= damage
    print("Hero defeats a troll, ")
    print("but takes", damage, "damage points.")

print("Hero lost but defeated ", trolls, " trolls.")
```

ANSWER - find the problem

health	health !=0
10	True
7	True
4	True
1	True
-2	True
-5 etc……	True

`health != 0` will never be true - <u>Infinite Loop</u>!

Rewrite condition: `while health >= 0`

While loop Summary

- Two common *while* loop errors:
  - Infinite Loops
  - Off-by-One Error

- We reviewed the following.
  - How to use sentinel values such as *negative numbers* to denote the end of a data set.
  - Using a Boolean to control the loop.
  - How to check for divide by zero to prevent errors.
  - How to use loops to sum and average entered values.