Looping

- Branching allows the program to execute certain statements if certain conditions are met;

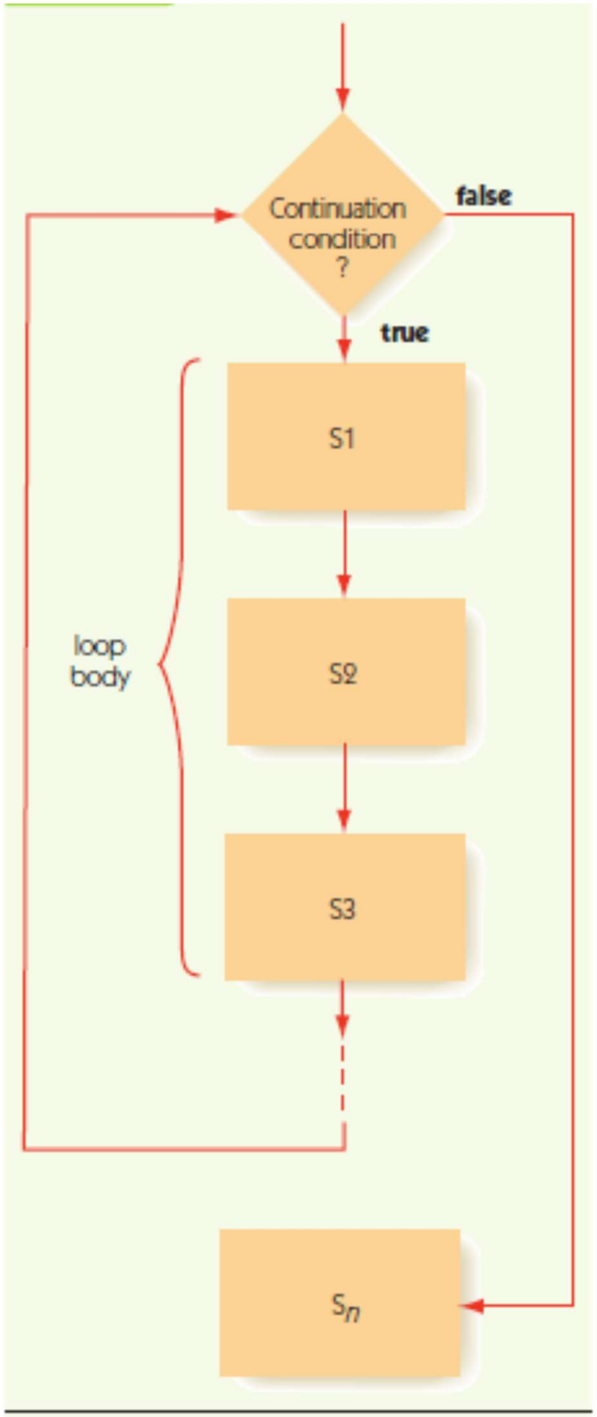
- We need to be able to repeat a block of instructions in a loop.

- Looping executes certain statements while certain conditions are met.

- We need a termination condition that tells us when to stop the loop.

- We may loop for a certain number of times, or until a certain condition occurs.

- As long as condition holds, the block of statements (loop body) will be repeatedly executed.
  - Otherwise, the statements are ignored.



Execution of the while loop

WHILE

- SYNTAX

```
WHILE "a true/false condition" remains true
        statement
        …
        statement
End of loop
```

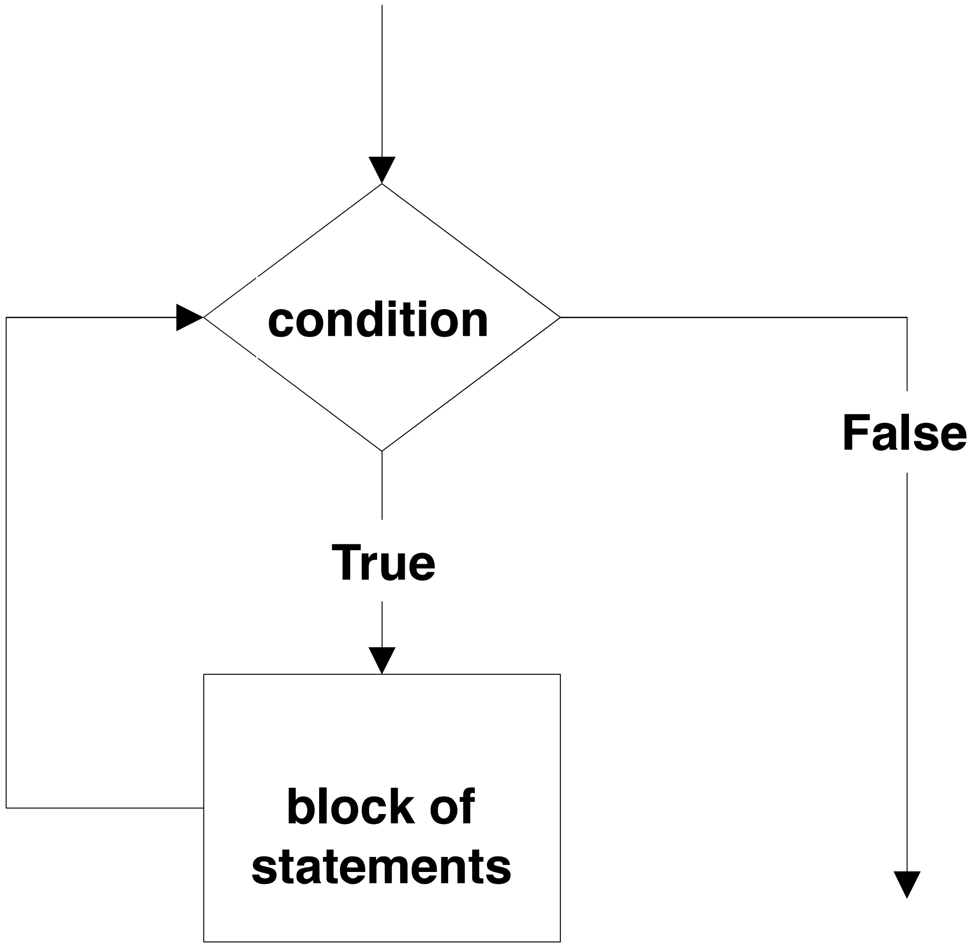
condition

loop body

- EXAMPLE

```
set the value of counter to 0
WHILE (counter < 100)
        statement
        …
        statement
        increment counter (add 1 to it)
End of loop
```

WHILE Loop Flow Chart



`while` loop example

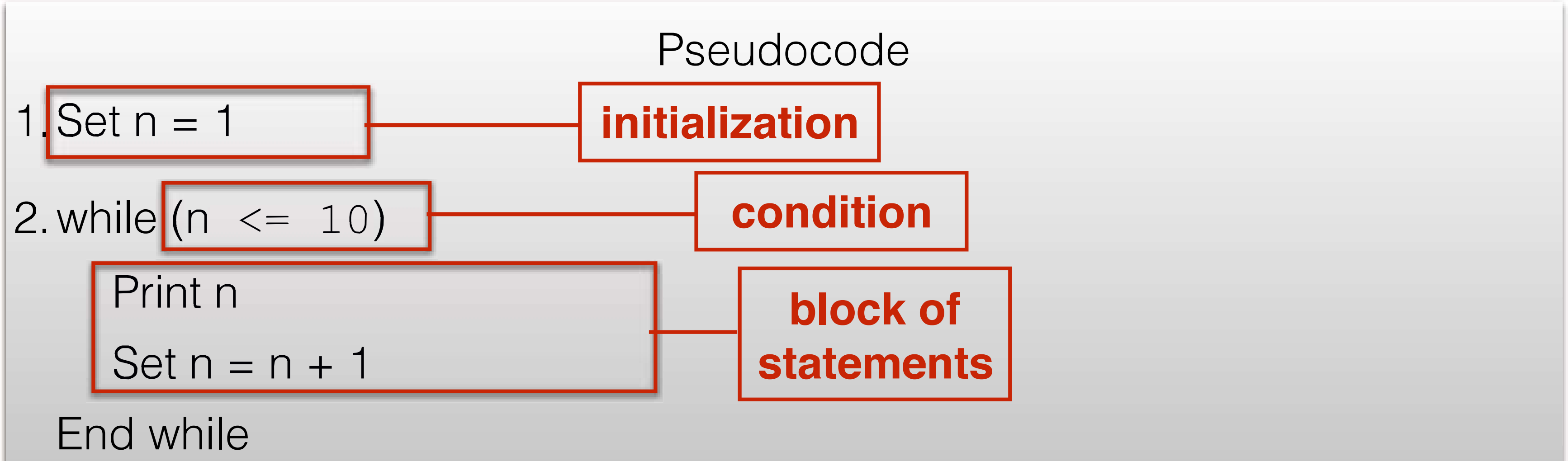
- Print the first 10 whole numbers.

Pseudocode

```
1. Set n = 1

2. while (n  <= 10)
       Print n
       Set n = n + 1
   End while
```

`while` loop example

- Print the first 10 whole numbers.



Pseudocode

```
1. Set n = 1
2. while (n <= 10)
       Print n
       Set n = n + 1
   End while
```

- **Each time we execute the body of the loop is called an *iteration*; e.g. this loop has 10 iterations**

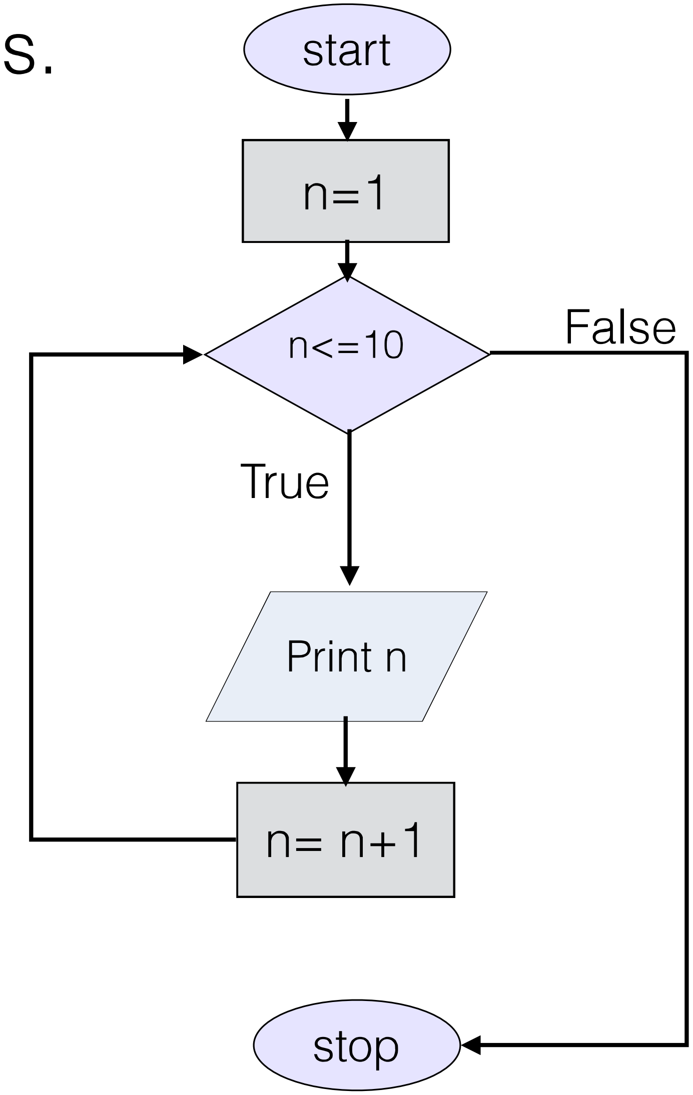
- **The body of the loop should change the value of one or more variables so that eventually the condition becomes false and the loop terminates.**

`while` loop example

- Print the first 10 whole numbers.

Pseudocode

```
1. Set n = 1
2. while (n <= 10)
       Print n
       Set n = n + 1
   End while
```



start

n=1

n<=10

False

True

Print n

n= n+1

stop

Designing Loops

- Designing a loop involves designing:
  - The body of the loop
  - The initialization
  - The conditions for ending the loop

Infinite Loops

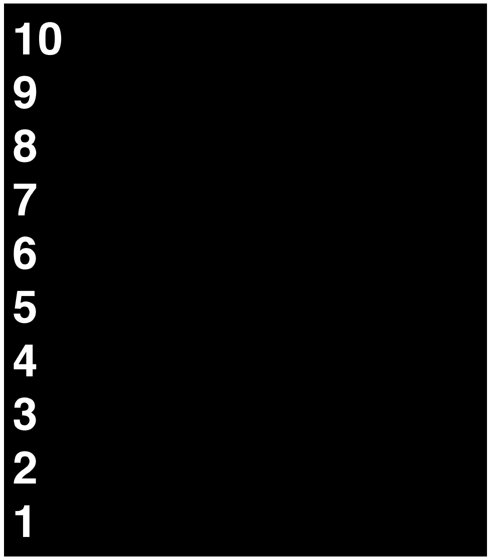
- Loops that never stop are infinite

Pseudocode

```
1. Set n = 1
2. While (n <= 10)
       Print n
   End while
```

`while` loop: more examples

- Print the first 10 whole numbers in reverse order



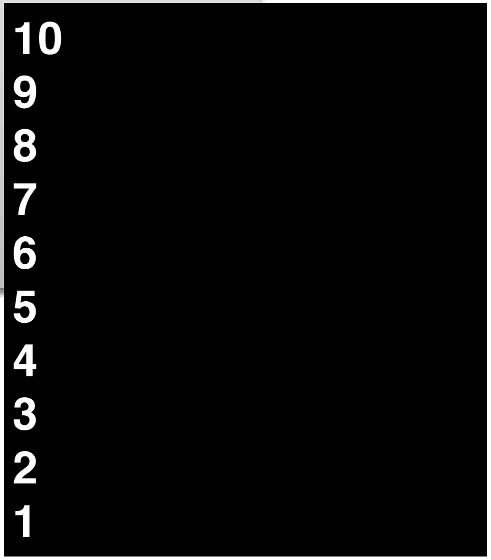
```
10
9
8
7
6
5
4
3
2
1
```

`while` loop: more examples

- Print the first 10 whole numbers in reverse order

Pseudocode

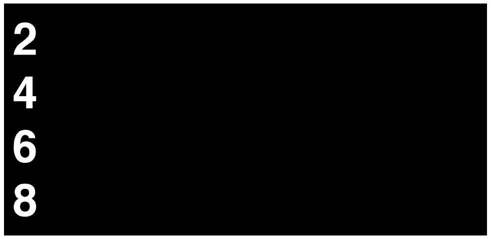
```
1. Set n = 10
2. While (n >=1)
       Print n
       Set n = n - 1
   End while
```



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

`while` loop: more examples

- Print the even numbers less than 10



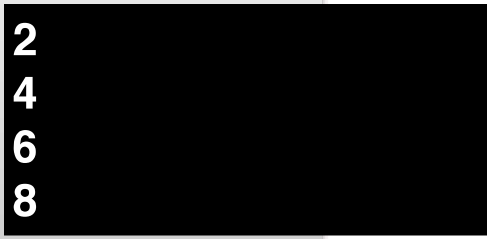
```
2
4
6
8
```

`while` loop: more examples

- Print the even numbers less than 10

Pseudocode

```
1. Set n = 2
2. While (n < 10)
       Print n
       Set n = n + 2
   End while
```



```
2
4
6
8
```

Looping

- Branching allows the program to execute certain statements if certain conditions are met;

- Loops execute certain statements while certain conditions are met.

- Python has two kinds of loops:
    - While
    - for

`while` - loop

- The while loop has a syntax similar to the if statement.

```
while condition:
   statement
   statement
   statement
```

```
if condition:
   statement
   statement
   statement
```

- As long as condition holds, the block of statements will be repeatedly executed.
  - Otherwise, the statements are ignored.

while - loop

- The while loop has a syntax similar to the if statement.

```
while condition:

   statement
   statement
   statement
```

while is a python keyword

- As long as condition holds, the block of statements will be repeatedly executed.
    - Otherwise, the statements are ignored.

`while` - loop

- The while loop has a syntax similar to the if statement.

```
while condition:
   statement
   statement
   statement
```

- **`condition` is boolean expression**
- **: is necessary**
- **Do not forget the indentation!**

- As long as condition holds, the block of statements will be repeatedly executed.
    - Otherwise, the statements are ignored.

`while` - loop

- The while loop has a syntax similar to the if statement.

```
while condition:

   statement
   statement
   statement
```

Flow of execution for a `while` statement:

1. **Evaluate the `condition`, yielding `True` or `False`.**
2. **If the `condition` is false, exit the `while` statement and continue execution at the next statement.**
3. **If the `condition` is true, execute the body and then go back to step 1.**

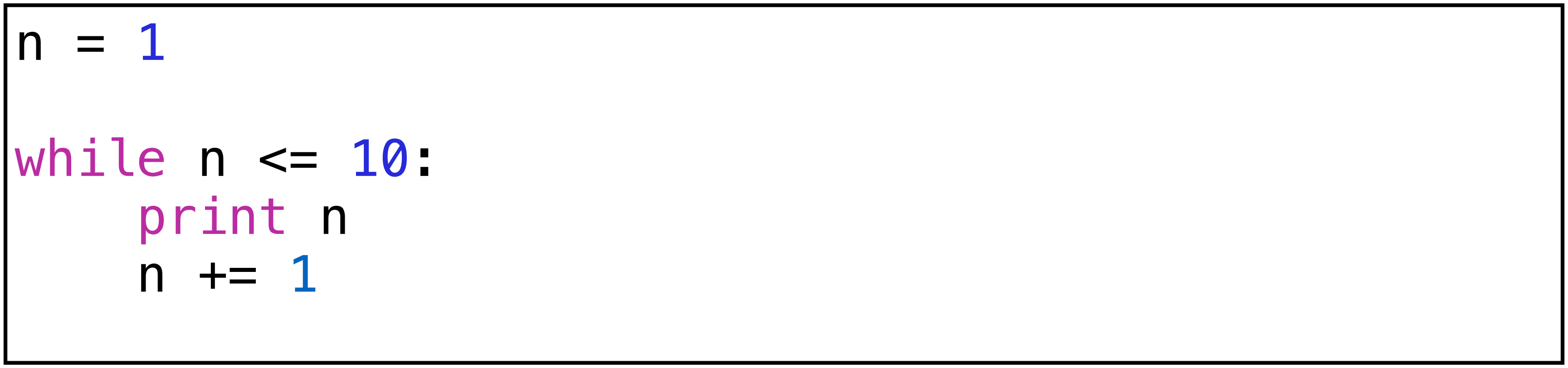
- As long as condition holds, the block of statements will be repeatedly executed.
    - Otherwise, the statements are ignored.

`while` loop example

- Print the first 10 whole numbers.

Pseudocode

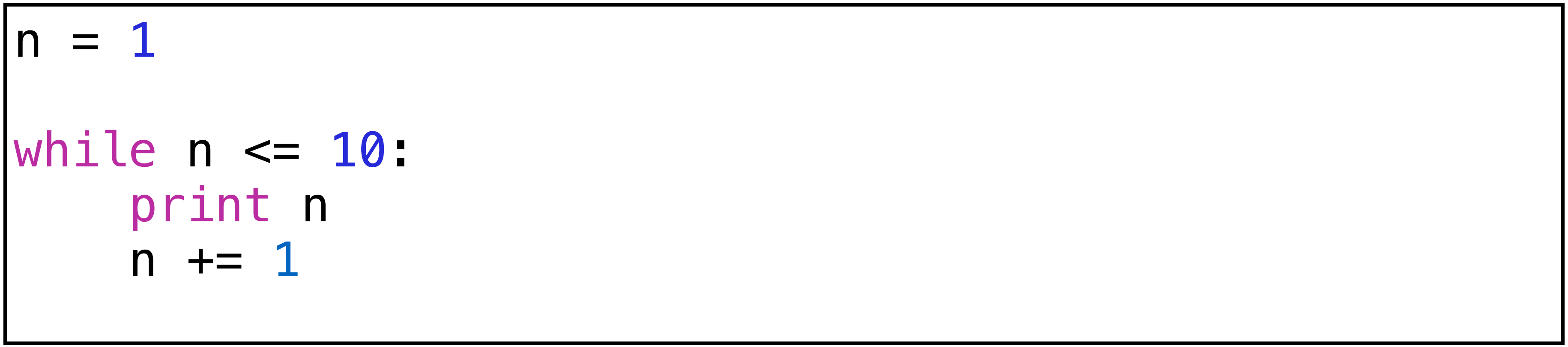
```
1. Set n = 1

2. while (n <= 10)
       Print n
       Set n = n + 1
   End while
```



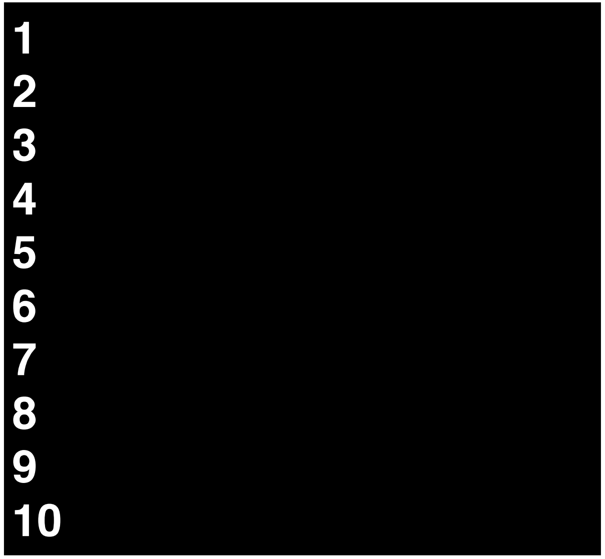
```
n = 1

while n <= 10:
    print n
    n += 1
```

`while` loop example

- Print the first 10 whole numbers.



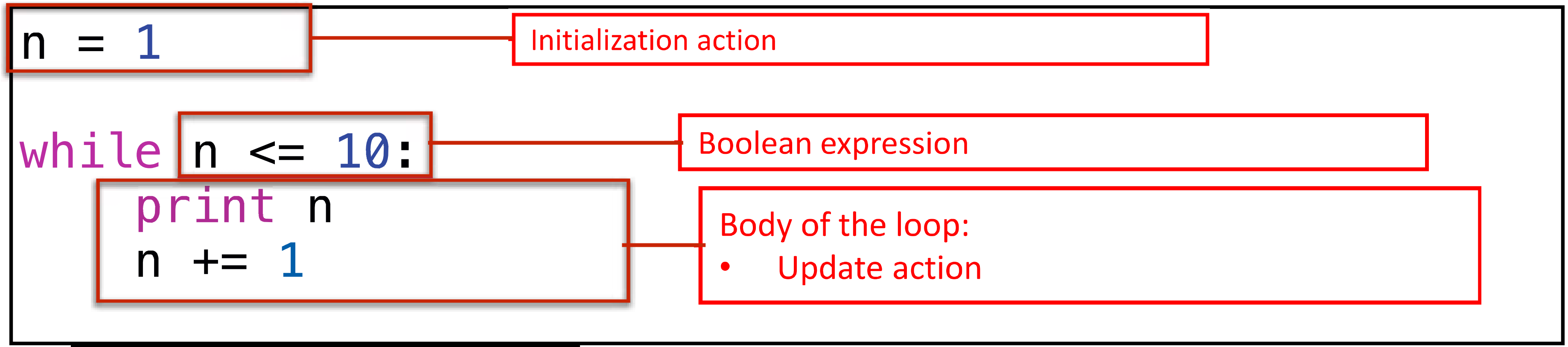
```
n = 1

while n <= 10:
    print n
    n += 1
```



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

while loop example

- Print the first 10 whole numbers.

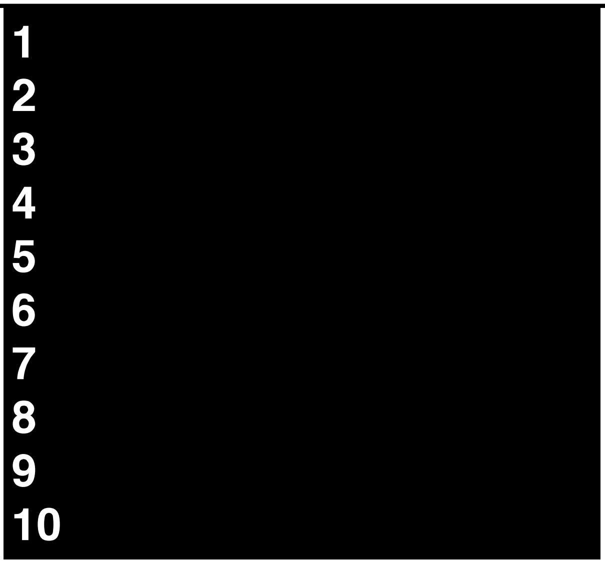


```
n = 1

while n <= 10:
    print n
    n += 1
```

Initialization action

Boolean expression

Body of the loop:
- Update action



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

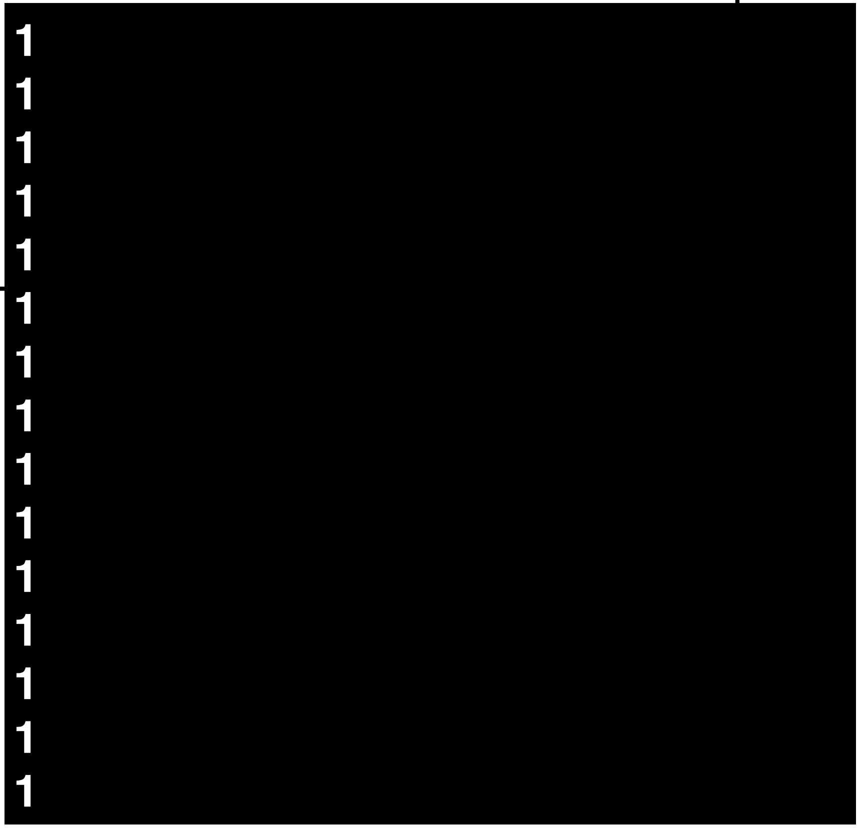
- **Each time we execute the body of the loop is called an *iteration*; e.g. this loop has 10 iterations**

- **The body of the loop should change the value of one or more variables so that eventually the condition becomes false and the loop terminates.**

Infinite Loops

- Loops that never stop are infinite

```
n = 1

while n <= 10:
    print n
    #n += 1
```



```
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
```

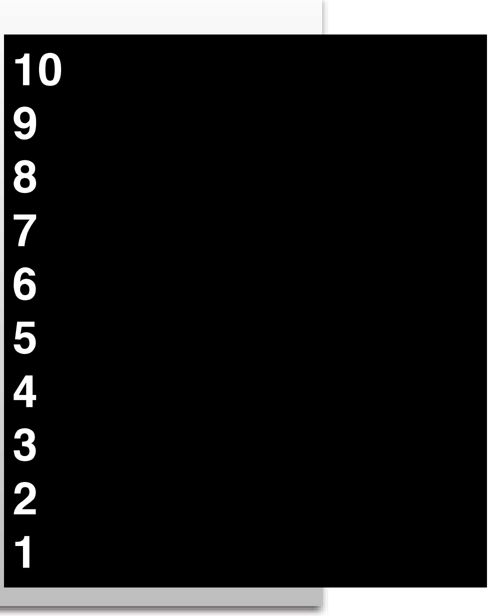
Terminate the program: ctrl+c

`while` loop: more examples

- Print the first 10 whole numbers in reverse order

Pseudocode

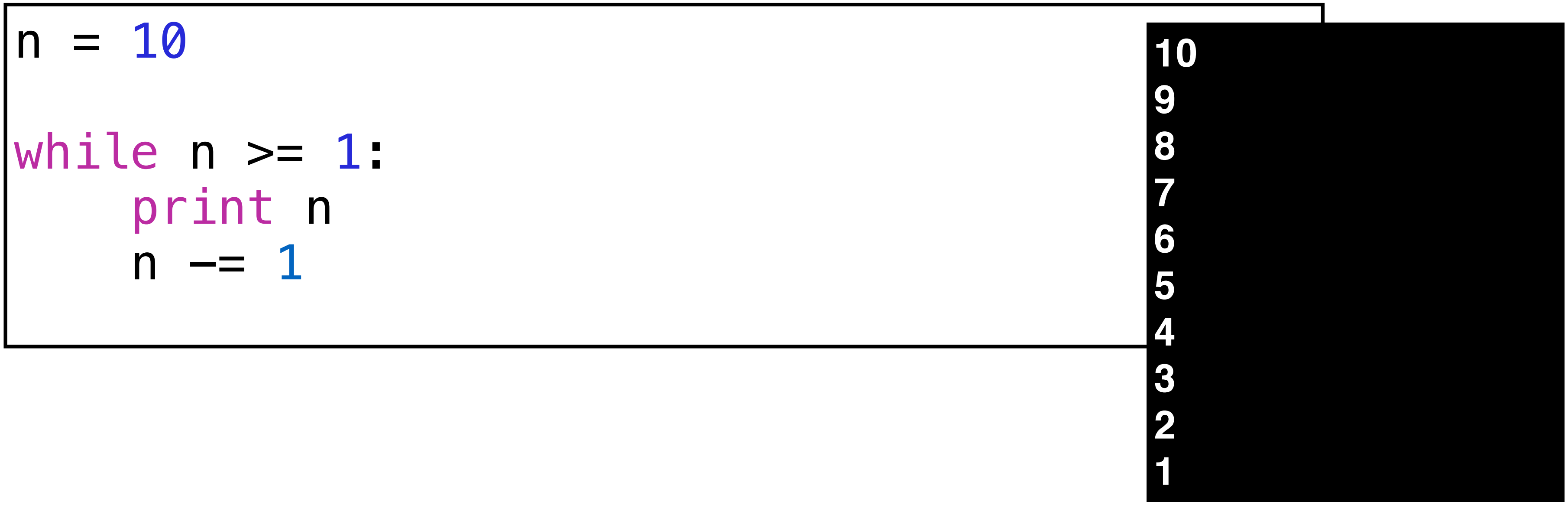
```
1. Set n = 10
2. While (n >=1)
       Print n
       Set n = n - 1
   End while
```



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

`while` loop: more examples

- Print the first 10 whole numbers in reverse order



```
n = 10

while n >= 1:
    print n
    n -= 1
```

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

`while` loop: more examples

- Print the even numbers less than 10

Pseudocode

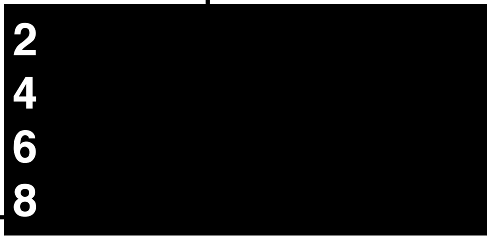
```
1. Set n = 2
2. While (n < 10)
       Print n
       Set n = n + 2
   End while
```

```
2
4
6
8
```

while loop: more examples

- Print the even numbers less than 10

```
n = 2

while n < 10:
    print n
    n += 2
```



```
2
4
6
8
```

while loop: more examples

- Print the first 10 even numbers

Pseudocode

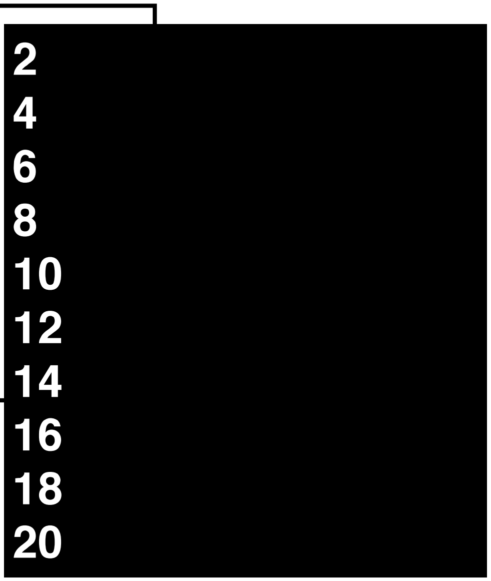
```
1. Set n = 2
2. Set counter = 1
3. While (counter <= 10)
       Print n
       Set n = n + 2
       Set counter = counter + 1
   End while
```

```
2
4
6
8
10
12
14
16
18
20
```

`while` loop: more examples

- Print the first 10 even numbers

```
n = 2
counter = 0

while counter < 10:
    print n
    n += 2
    counter += 1
```



```
2
4
6
8
10
12
14
16
18
20
```

`while` loop: more examples

- Write a program to compute and print the sum: 1+2+3+…+100
- Write a pseudocode and flow-chart first

`while` loop: more examples

- Compute and print the sum: 1+2+3+…+100

Pseudocode

```
1. Set n = 1
2. Set Sum = 0
3. While (n <= 100)
       Set Sum = Sum + n
       Set n = n + 1
   End while
4. Print "Sum of 1 to 100 is:" , Sum
```

`while` loop: more examples

- Compute and print the sum: 1+2+3+…+100

```python
n = 1
sum = 0

while n <= 100:
    sum += n
    n += 1

print "sum:", sum
```

```
sum: 5050
```

Looping

- What will be the output of the following code?

A. hi
hi

B. hi hi hi

C. hi
hi
hi

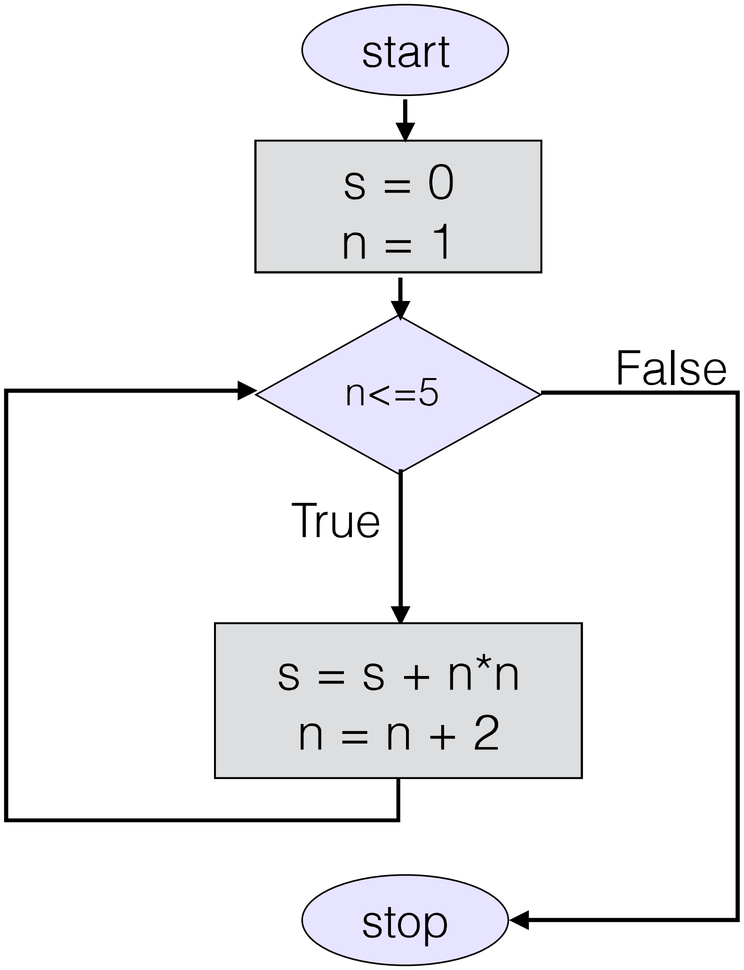
D. hi hi

E. hi

```
n = 0

while n <= 2:
    print 'hi'
    n += 1
```

Looping

- What will be the value of `sum` at the stop point of this flow-chart?



A. $1^2 + 2^2 + 3^2$

B. $1^2 + 2^2 + 3^2 + 4^2$

C. $1^2 + 3^2 + 5^2$

D. $1^2 + 3^2 + 5^2 + 7^2$

E. $1^2 + 3^2$

Looping

- What will be the value of `n` immediately after executing this code?

A. 5

B. 6

C. 7

D. 8

E. 9

```
n = 1
sum = 0

while n <= 5:
    sum += n*n
    n += 2
```

More Examples

- Write a code to compute n!
    - for example: 5! = 5*4*3*2*1 = 120

    - Draw a flow-chart for it

More Examples

- Write a code to compute n!
  - 5! = 5*4*3*2*1 = 120

```
i = 1
prod = 1
n = raw_input("Enter a positive integer: “)
n = int(n)        # convert input string to int

while i <= n:
    prod *= i
    i += 1
print n,’! is’, prod
```

```
5! is 120
```

Looping

- What is the value of `prod` immediately after executing this code?

```
n = 5

while n >= 1:
    prod *= n
    n -= 1
```

(A)120

(B) 24

(C) 0

(D) Cannot be determined!

(E) Won’t compile!

`break` Statement

- `break` statement terminates the loop (exit loop or jump out of loop) and control flow moves to the statement after the the loop.
- `break` may only occur syntactically nested in a loop
- Example:
  - Sometimes you don't know it's time to end a loop until you get half way through the body.
  - In that case you can write an infinite loop on purpose and then use the `break` statement to jump out of the loop.

`break` Statement

- For example, suppose you want to take input from the user and print it in the output until they type “done”.

```
while True:
    line = raw_input("> ")
    if line == 'done':
        break
    print line

print "Done!!"
```

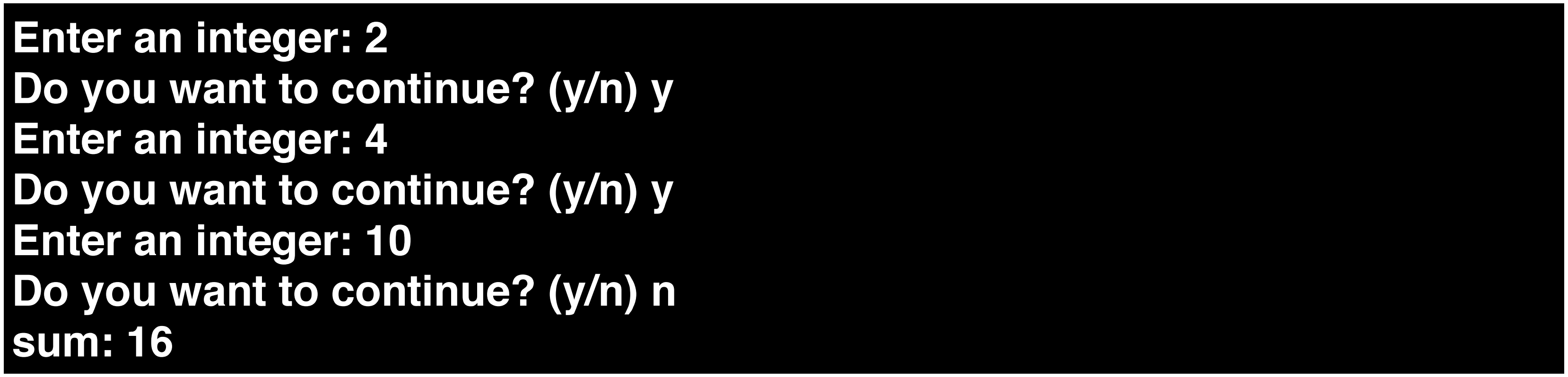
break Statement

- For example, suppose you want to take input from the user and print it in the output until they type “done”.

```
while True:
    line = raw_input("> ")
    if line == 'done':
        break
    print line

print "Done!!"
```

```
> Hey
Hey
> Hello!
Hello!
> I am done!
I am done!
> done
done
Done!!
```

- Write a program to compute the summation of some numbers entered by the users (you should ask the user to enter the number and continue as long as the user wish)



```
Enter an integer: 2
Do you want to continue? (y/n) y
Enter an integer: 4
Do you want to continue? (y/n) y
Enter an integer: 10
Do you want to continue? (y/n) n
sum: 16
```

- Write a program to compute the summation of some numbers entered by the users (you should ask the user to enter the number and continue as long as the user wish)

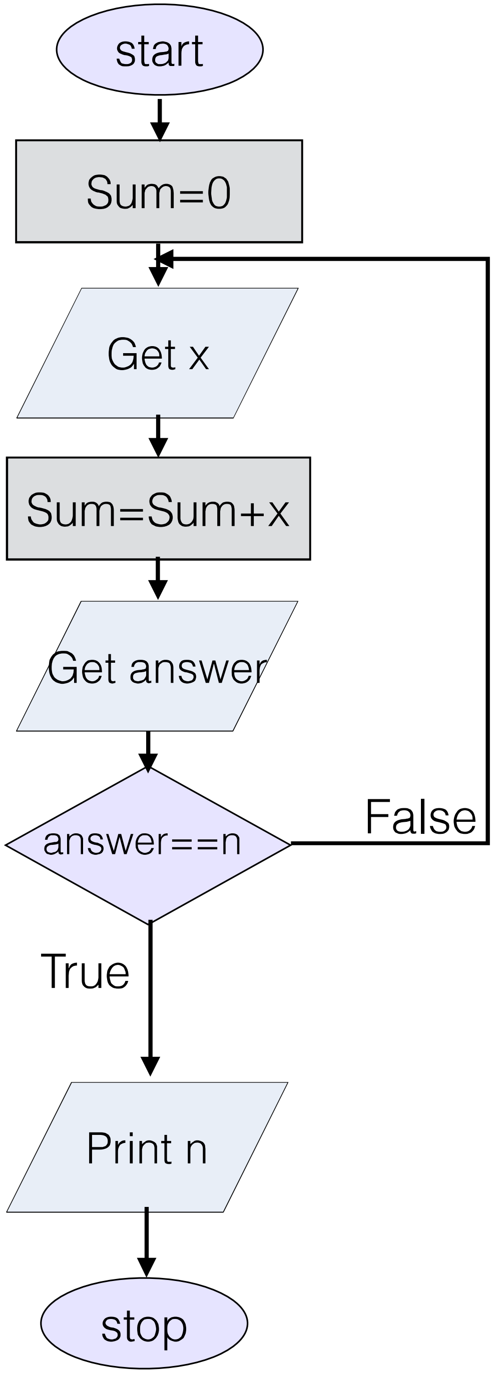
Pseudocode

```
1. Set Sum = 0

2. While (True)
       Get x
       Set Sum = Sum + x
       Print “Do you want to continue?(y/n) ”
       Get  answer
       If (answer==“n”)
           Go out of loop
   End while

3. Print “Sum of numbers is:” , Sum
```

- Write a program to compute the summation of some numbers entered by the users (you should ask the user to enter the number and continue as long as the user wish)

```
sum = 0
while True:
    x = raw_input("Enter an integer: ")
    x = int(x)
    sum += x

    answer = raw_input("Do you want to continue? (y/n) ")
    if answer == 'n':
        break

print "sum:", sum
```

```
Enter an integer: 2
Do you want to continue? (y/n) y
Enter an integer: 4
Do you want to continue? (y/n) y
Enter an integer: 10
Do you want to continue? (y/n) n
sum: 16
```


start
Sum=0
Get x
Sum=Sum+x
Get answer
answer==n
False
True
Print n
stop

- Write a program to compute the summation of some numbers entered by the users (you should ask the user to enter the number and continue as long as the user wish)

Practice: Change this code, in a way that would not be case sensitive

```
sum = 0
while True:
    x = raw_input("Enter an integer: ")
    x = int(x)
    sum += x

    answer = raw_input("Do you want to continue? (y/n) ")
    if answer == 'n':
        break

print "sum:", sum
```

```
Enter an integer: 2
Do you want to continue? (y/n) y
Enter an integer: 4
Do you want to continue? (y/n) y
Enter an integer: 10
Do you want to continue? (y/n) n
sum: 16
```

Looping

- What will be the value of `sum` immediately after executing this code?

$A.\ \ 1^2 + 2^2$

$B.\ \ 1^2 + 2^2 + 3^2$

$C.\ \ 1^2 + 3^2 + 5^2$

$D.\ \ 1^2 + 3^2$

$E.\ \ 1^2$

```
n = 1
sum = 0

while n <= 5:
    sum += n*n
    n += 2
    if n == 3:
        break
```

Looping

- What will be the value of `sum` immediately after executing this code?

$A.\ \ 1^2 + 2^2$

$B.\ \ 1^2 + 2^2 + 3^2 + 4^2$

$C.\ \ 1^2 + 3^2 + 5^2$

$D.\ \ 1^2 + 3^2 + 5^2 + 7^2$

$E.\ \ 1^2 + 3^2$

```
n = 1
sum = 0

while n <= 5:
    sum += n*n
    n += 2
    if n == 4:
       break
```

Looping

- What will be the value of `sum` immediately after executing this code?

$A.\ \ 1^2 + 2^2$

$B.\ \ 1^2 + 2^2 + 3^2$

$C.\ \ 1^2 + 3^2 + 5^2$

$D.\ \ 1^2 + 3^2$

$E.\ \ 1^2$

```
n = 1
sum = 0

while n <= 5:
    sum += n*n
    if n == 3:
       break
    n += 2
```

`continue` Statement

- `continue` statement terminates ***the current iteration*** and immediately goes to the next iteration (it <u>does not</u> terminate the loop!!)
- `continue` may only occur syntactically nested in a loop
- Example:
    - Sometimes (if some specific condition holds) you want to skip to the next iteration without finishing the body of the loop for the current iteration
    - In that case you can use `continue` statement

- Write a program to compute the summation of some positive numbers entered by the users (you should ask the user to enter the number and continue as long as the user wish). If the user enters a negative number you should print a message for the user that the input is invalid and ignore that input and continue

```
Enter an integer: 2
Do you want to continue? (y/n) y
Enter an integer: -4
Do you want to continue? (y/n) y
Enter an integer: 10
Do you want to continue? (y/n) n
sum: 8
```

```
Enter an integer: 2
Do you want to continue? (y/n) y
Enter an integer: -4
invalid input! Please try again!
Enter an integer: 10
Do you want to continue? (y/n) n
sum: 12
```

- Write a program to compute the summation of some positive numbers entered by the users (you should ask the user to enter the number and continue as long as the user wish). If the user enters a negative number you should print a message for the user that the input is invalid and ignore that input and continue

This is our previous pseudocode, we need to update it

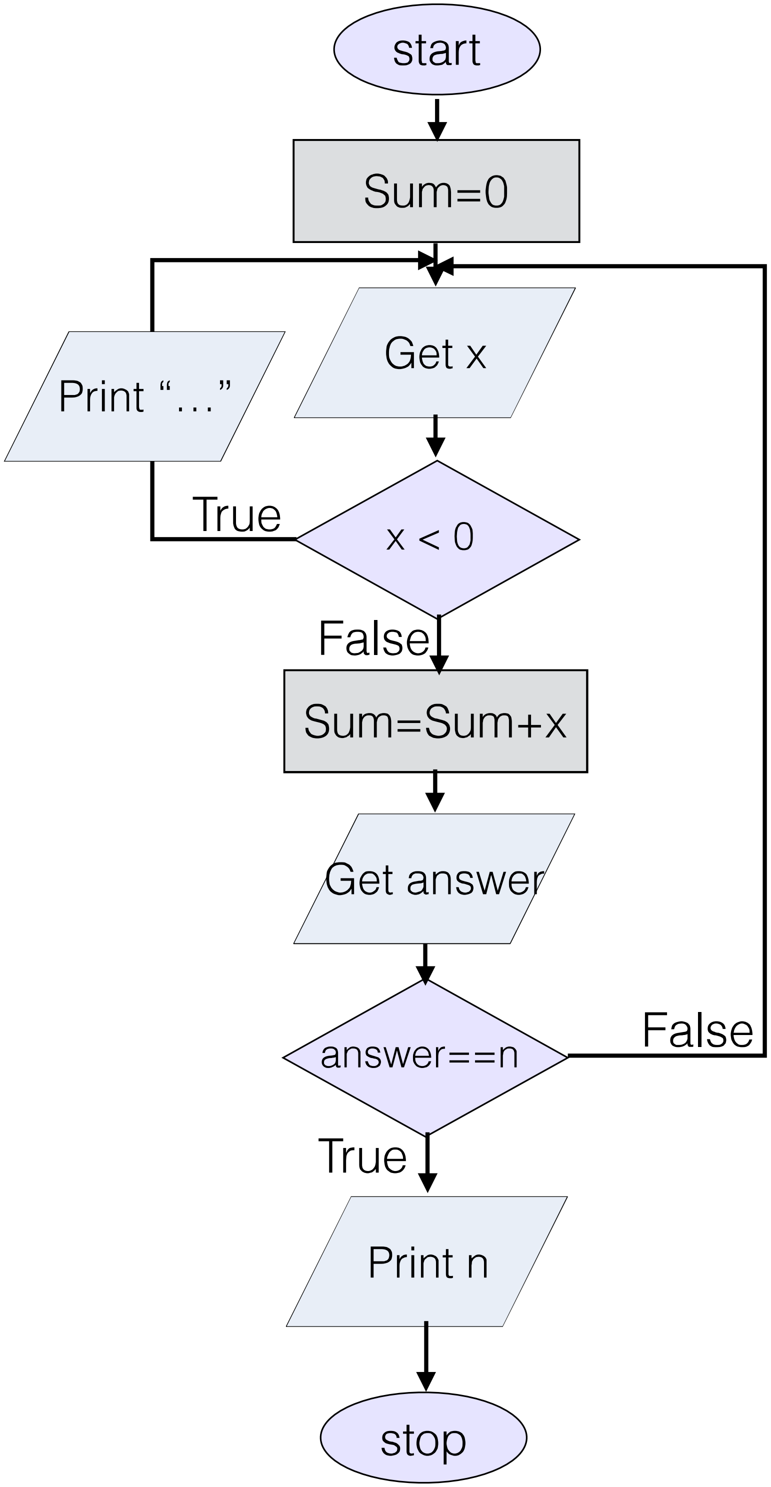
Pseudocode

```
1. Set Sum = 0

2. While (True)
       Get x
       Set Sum = Sum + x
       Print “Do you want to continue?(y/n) ”
       Get answer
       If (answer==“n”)
           Go out of loop
   End while

3. Print “Sum of numbers is:” , Sum
```

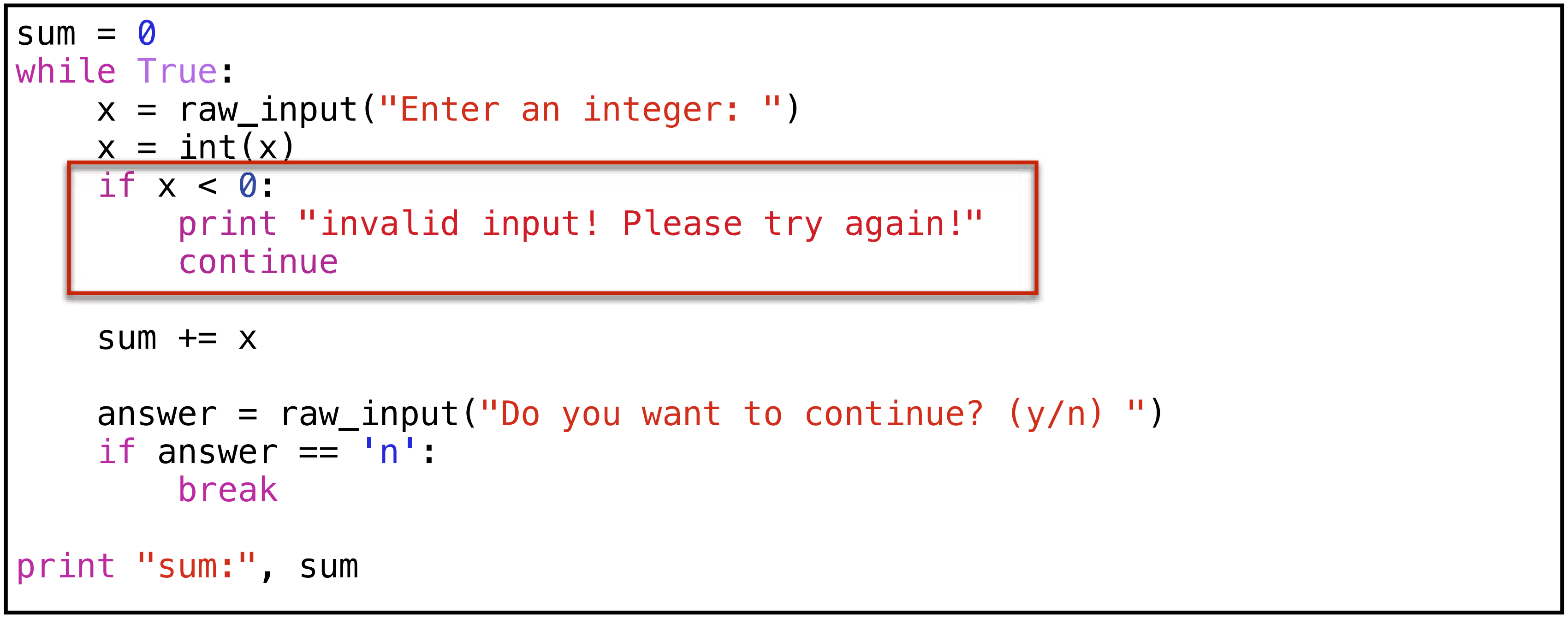
- Write a program to compute the summation of some positive numbers entered by the users (you should ask the user to enter the number and continue as long as the user wish). If the user enters a negative number you should print a message for the user that the input is invalid and ignore that input and continue

Pseudocode

```
1. Set Sum = 0
2. While (True)
      Get x
      If (x < 0)
          Go to the beginning of the loop          #ignore the rest of the loop
      Set Sum = Sum + x
      Print “Do you want to continue?(y/n) ”
      Get answer
      If (answer==“n”)
          Go out of loop
   End while
3. Print “Sum of numbers is:” , Sum
```


start
Sum=0
Get x
Print "…"
True
x < 0
False
Sum=Sum+x
Get answer
answer==n
False
True
Print n
stop

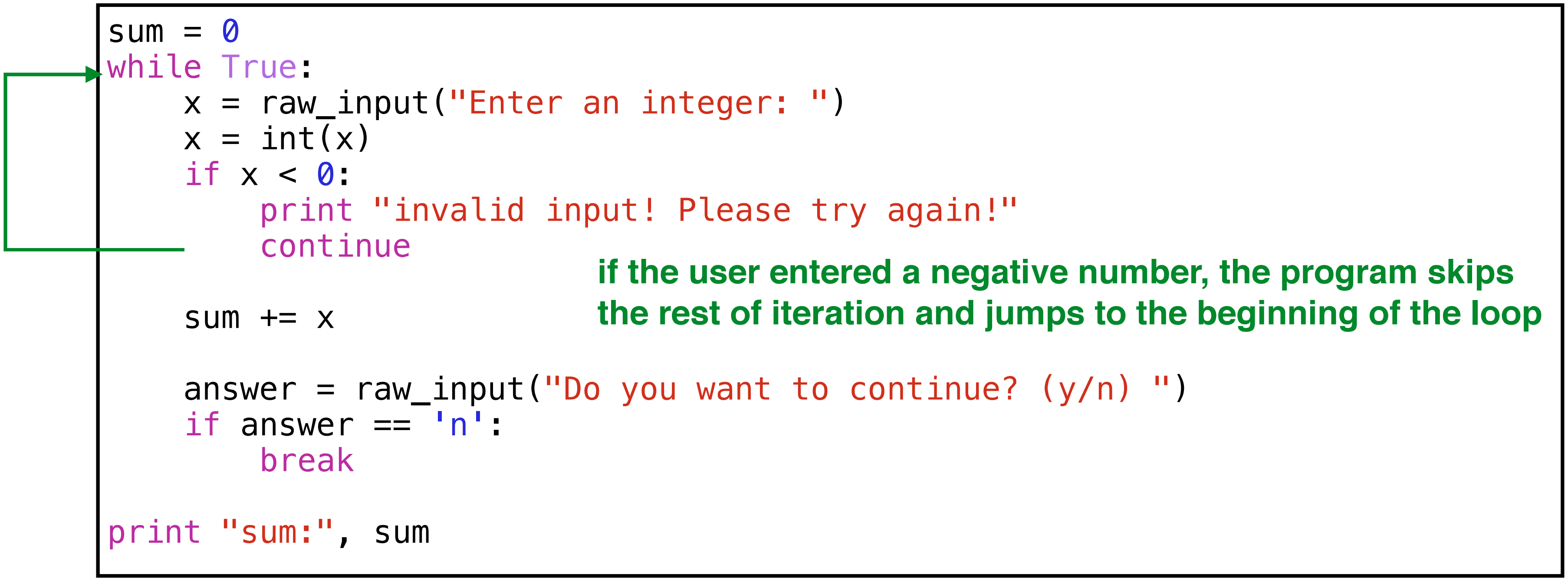
- Write a program to compute the summation of some positive numbers entered by the users (you should ask the user to enter the number and continue as long as the user wish). If the user enters a negative number you should print a message for the user that the input is invalid and ignore that input and continue



```
sum = 0
while True:
    x = raw_input("Enter an integer: ")
    x = int(x)
    if x < 0:
        print "invalid input! Please try again!"
        continue

    sum += x

    answer = raw_input("Do you want to continue? (y/n) ")
    if answer == 'n':
        break

print "sum:", sum
```

```
Enter an integer: 2
Do you want to continue? (y/n) y
Enter an integer: -4
invalid input! Please try again!
Enter an integer: 10
Do you want to continue? (y/n) n
sum: 12
```

- Write a program to compute the summation of some positive numbers entered by the users (you should ask the user to enter the number and continue as long as the user wish). If the user enters a negative number you should print a message for the user that the input is invalid and ignore that input and continue



```
sum = 0
while True:
    x = raw_input("Enter an integer: ")
    x = int(x)
    if x < 0:
        print "invalid input! Please try again!"
        continue

    sum += x

    answer = raw_input("Do you want to continue? (y/n) ")
    if answer == 'n':
        break

print "sum:", sum
```

if the user entered a negative number, the program skips the rest of iteration and jumps to the beginning of the loop

```
Enter an integer: 2
Do you want to continue? (y/n) y
Enter an integer: -4
invalid input! Please try again!
Enter an integer: 10
Do you want to continue? (y/n) n
sum: 12
```

More Examples

- Write a Python code which asks user to enter a positive integer number and prints the digits of this number (one per line)

More Examples

- Write a Python code which asks user to enter a positive integer number and prints the digits of this number (one per line)

Pseudocode

```
Get number
While (number > 0)
    Set digit = number % 10
    Print digit
    Set number = number / 10
End while
```

More Examples

- Write a Python code which asks user to enter a positive integer number and prints the digits of this number (one per line)

```
number = raw_input("Enter an integer number: ")
number = int(number)       # convert input string to int

while number > 0:
    digit = number % 10
    print digit
    number /= 10

print "done!"
```

More Examples

- Write a Python code which asks user to enter a positive integer number and prints the digits of this number (one per line)

```
number = raw_input("Enter an integer number: ")
number = int(number)        # convert input string to int

while number > 0:
    digit = number % 10
    print digit
    number /= 10

print "done!"
```

Practice: Change this code to compute and print the summation of digits instead of printing them. For example, for input: 3257, the output is: 17

More Examples

- Loops are useful for implementing sequences and series.
- A classic example is the Fibonacci sequence:

  1,1,2,3,5,8,13,21,34,55,89,144,…

- In the Fibonacci sequence, each number is found by adding up the two numbers before it, except for the first two terms that are set to be equal to 1. In other words, the sequence is defined by:

$$F_n = F_{n-1} + F_{n-2}$$

- given

$$F_1 = F_2 = 1$$

More Examples

$$F_n = F_{n-1} + F_{n-2}$$

$$F_1 = F_2 = 1$$

- How many variables do we need?

More Examples

$$F_n = F_{n-1} + F_{n-2}$$

$$F_1 = F_2 = 1$$

- How many variables do we need?
- To compute the next number in the sequence we just need the two previous ones
  - In total, we just need 3 distinct variables to keep: F_n , F_{n-1} , F_{n-2}

More Examples

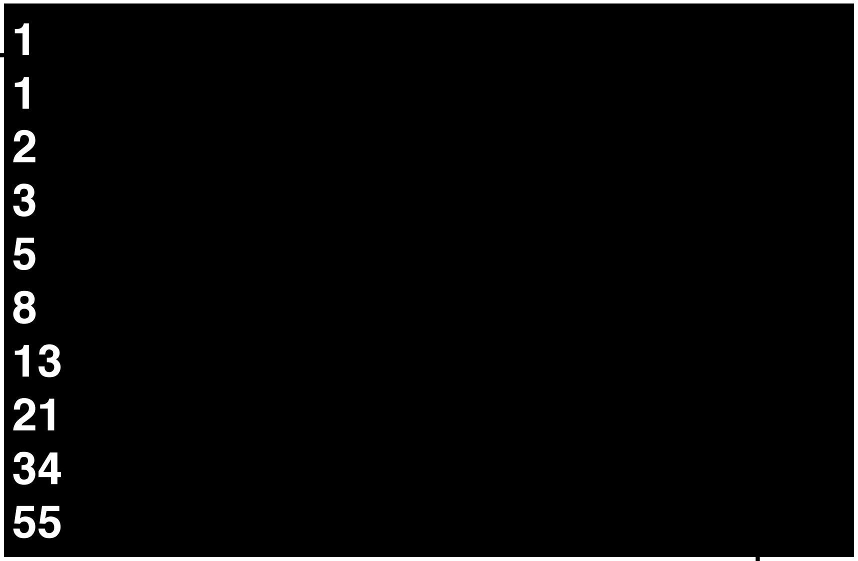
- Write a program to print the first 10 Fibonacci numbers:

```
F1 = 1
F2 = 1
counter = 3
print F1
print F2

while counter <= 10:
    F3 = F1 + F2
    print F3
    F1 = F2
    F2 = F3
    counter += 1
```

More Examples

- Write a program to print the first 10 Fibonacci numbers:

```
F1 = 1
F2 = 1
counter = 3
print F1
print F2

while counter <= 10:
    F3 = F1 + F2
    print F3
    F1 = F2
    F2 = F3
    counter += 1
```



```
1
1
2
3
5
8
13
21
34
55
```

Review

- Each pass through a loop is called ...

A. Pass through

B. Enumeration

C. Culmination

D. Iteration

Review

- A break statement causes execution to skip to the ...

A. next iteration of the loop.

B. first statement after the loop.

C. end of the program.

Debugging Loops

- Common errors involving loops include
  - Off-by-one errors in which the loop executes one too many or one too few times
    - Look at Task-1 in Lab-3
  - Infinite loops usually result from a mistake in the Boolean expression that controls the loop
    - Look at Task-2 in Lab-3
- Advice for debugging: be sure that the mistake is really in the loop; trace the variable to observe how their values change through out the loop.

Type of Errors

- Syntax Errors
- Run-time Errors
- Logical Errors
- Numerical Errors
  - When using numerical methods or algorithms and computing with finite precision, errors of approximation or rounding and truncation are introduced.

Numerical Errors

- Rounding (round-off) error
    - occurs because of the computing device's inability to deal with certain numbers. Such numbers need to be rounded off to some near approximation which is dependent on the memory size used to represent numbers of the device.
    - https://en.wikipedia.org/wiki/Round-off_error

Rounding Error

- Why the following code results in an infinite loop?

```
x = 0.1
while x!=0.2:
    x += 0.001

print "done!"
```

Suggested Reading

- Chapter 5 (except `for` loop)