

# 6.1 Loops (general)

## Loop concept

People who have children may be familiar with looping around the block until a baby falls asleep.

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

**PARTICIPATION ACTIVITY**

6.1.1: Loop concept: Driving a baby around the block.



### Animation captions:

1. Parents may be familiar with this scenario: Driving home, baby is awake. Parents circle the block, hoping the baby will fall asleep.
2. After first loop, baby is still awake, so parents loop again.
3. After second loop, baby is asleep, so parents head home for a peaceful evening.

**PARTICIPATION ACTIVITY**

6.1.2: Loop concept.



Consider the example above.

- 1) When the parents first checked, was the baby awake?

- Yes
- No



- 2) After the first loop, was the baby awake?

- Yes
- No



- 3) After the second loop, was the baby awake?

- Yes
- No

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4



- 4) How many loops around the block did the parents make?

- 2



3

- 5) Where was the decision point for whether to loop: At the top of the street or bottom?

Top  
 Bottom

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4



## Loop basics

A **loop** is a program construct that repeatedly executes the loop's statements (known as the **loop body**) while the loop's expression is true; when false, execution proceeds past the loop. Each time through a loop's statements is called an **iteration**.

PARTICIPATION ACTIVITY

6.1.3: A simple loop: Summing the input values.



### Animation captions:

1. A loop is like a branch, but jumping back to the expression when done. Thus, the loop's statements may execute multiple times, before execution proceeds past the loop.
2. This program gets an input value. If the value  $> -1$ , the program adds the value to a sum, gets another input, and repeats. val is 2, so the loop's statements execute, making sum 2.
3. The loop's statements ended by getting the next input, which is 4. The loop's expression  $4 > -1$  is true, so the loop's statements execute again, making sum  $2 + 4$  or 6.
4. The loop's statements got the next input of 1. The loop's expression  $1 > -1$  is true, so the loop's statements execute a third time, making sum  $6 + 1$  or 7.
5. The next input is -1. This time,  $-1 > -1$  is false, so the loop is not entered. Instead, execution proceeds past the loop, where a statement puts sum, which is 7, to the output.

## Loop example: Computing an average

A loop can be used to compute the average of a list of numbers.

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

PARTICIPATION ACTIVITY

6.1.4: Loop example: Computing an average.



### Animation captions:

1. The program computes an average of a list of numbers (a negative ends the list). The first input is 2, so the loop is entered. Sum becomes 2, and num is incremented to 1.

2. The next input is 4. The loop is entered, so sum becomes  $2 + 4$  or 6, and num is incremented to 2.
3. The next input is 9, so the loop is entered. Sum becomes  $6 + 9$  or 15, and num is incremented to 3.
4. The next input is -1, so the loop is not entered.  $15 / 3$  or 5 is output.

**PARTICIPATION ACTIVITY****6.1.5: Loop example: Average.**

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4



Consider the computing an average example above.

- 1) In the example above, the first value gotten from input was 2. That caused the loop body to be \_\_\_\_.

 executed not executed

- 2) At the end of the loop body, the \_\_\_\_.

 next input is gotten loop is exited average is computed

- 3) With what value was sum initialized?

 -1 0

- 4) Each time through the loop, the sum variable is increased by \_\_\_\_.

 0 1 the current input value

- 5) What was variable num's value after the loop was done iterating?

 1 2 3

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4



- 6) Before the loop, the first input value is



gotten. If that input was negative (unlike the data in the example above), the loop's body would \_\_\_\_.

be executed

not be executed

## Example: Counting specific values in a list

©zyBooks 09/27/22 11:37 469702

Steven Cameron

WGUC859v4

Programs execute one statement at a time. Thus, using a loop to examine a list of values one value at a time and updating variables along the way, as in the above examples, is a common programming task.

Below is a task to help a person get accustomed to examining a list of values one value at a time. The task asks a person to count the number of negative values, incrementing a variable to keep count.

### PARTICIPATION ACTIVITY

6.1.6: Counting negative values in a list of values.



Click "Increment" if a negative value is seen.

Start

Counter

0



Next value

Increment

Time - Best time -

Clear best

### PARTICIPATION ACTIVITY

6.1.7: Counting negative values.

©zyBooks 09/27/22 11:37 469702

Steven Cameron

WGUC859v4



Complete the program such that variable count ends having the number of negative values in an input list of values (the list ends with 0). So if the input is -1 -5 9 3 0, then count should end with 2.

```
count = 0
val = Get next input

While val is not 0
    if __(A)__
        __(B)__

    val = Get next input
```

1) What should expression (A) be? ©zyBooks 09/27/22 11:37 469702

- val > 0
- val < 0
- val is 0

2) What should statement (B) be? Steven Cameron  
WGUC859v4

- val = val + 1
- count = count + 1
- count = val

3) If the input value is 0, does the loop body execute? WGUC859v4

- Yes
- No

## Example: Finding the max value

Examining items one at a time and updating a variable can achieve some interesting computations. The task below is to find the maximum value in a list of positive values. A variable stores the max value seen so far. Each input value is compared with that max, and if greater, that value replaces that max. The max value is initialized with -1 so that such comparison works even for the first input value.

**PARTICIPATION ACTIVITY**

6.1.8: Find the maximum value in the list of values. WGUC859v4

Click "Store value" if a new maximum value is seen. ©zyBooks 09/27/22 11:37 469702

Steven Cameron  
WGUC859v4

**Start**

**max**

X	X	X	X	X	X	X
---	---	---	---	---	---	---

[Next value](#)[Store value](#)

Time - Best time -

[Clear best](#)**PARTICIPATION ACTIVITY**

6.1.9: Determining the max value.

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

Complete the program such that variable max ends having the maximum value in an input list of positive values (the list ends with 0). So if the input is 22 5 99 3 0, then max should end as 99.

```
max = -1
val = Get next input

while val is not 0
    If __(A)__
        __(B)__

    val = Get next input
```

1) What should expression (A) be?

- max > 0
- max > val
- val > max

2) What should statement (B) be?

- max = val
- val = max
- max = max + 1

3) Does the final value of max depend on the order of inputs? In particular, would max be different for inputs 22 5 99 3 0 versus inputs 99 3 5 22 0?

- Yes
- No

4) For inputs 5 10 7 20 8 0, with what values should max be assigned?

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

O -1, 20

## 6.2 While loops

### While loop: Basics

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

A **while loop** is a construct that repeatedly executes an indented block of code (known as the **loop body**) as long as the loop's expression is True. At the end of the loop body, execution goes back to the while loop statement and the loop expression is evaluated again. If the loop expression is True, the loop body is executed again. But, if the expression evaluates to False, then execution instead proceeds to below the loop body. Each execution of the loop body is called an **iteration**, and looping is also called *iterating*.

Construct 6.2.1: While loop.

```
while expression: # Loop expression
    # Loop body: Sub-statements to execute
    # if the loop expression evaluates to True

    # Statements to execute after the expression evaluates to
    # False
```

PARTICIPATION  
ACTIVITY

6.2.1: While loop.



### Animation content:

undefined

### Animation captions:

1. When encountered, a while loop's expression is evaluated. If true, the loop's body is entered. Here, user\_char was initialized with 'y', so user\_char == 'y' is true. WGUC859v4
2. Thus, the loop body is executed, which outputs curr\_power's current value of 2, doubles curr\_power, and gets the next input.
3. Execution jumps back to the while part. user\_char is 'y' (the first input), so user\_char == 'y' is true, and the loop body executes (again), outputting 4.
4. user\_char is 'y' (the second user input), so user\_char == 'y' is true, and the loop body executes (a third time), outputting 8.

5. user\_char is now 'n', so user\_char == 'y' is false. Thus, execution jumps to after the loop, which outputs "Done".

**PARTICIPATION ACTIVITY****6.2.2: Basic while loops.**

For each question, indicate how many times will the loop body execute?

- 1) 

```
x = 3
while x >= 1:
    # Do something
    x = x - 1
```

**Check****Show answer**

- 2) Assume user would enter 'n', then 'n', then 'y'.

```
# Get character from user here
while user_char != 'n':
    # Do something
    # Get character from user
here
```

**Check****Show answer**

- 3) Assume user would enter 'a', then 'b', then 'n'.

```
# Get character from user here
while user_char != 'n':
    # Do something
    # Get character from user
here
```

**Check****Show answer**

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

## Example: While loop with a sentinel value

The following example uses the statement `while user_value != 'q':` to allow a user to end a face-drawing program by entering the character 'q'. The letter 'q' in this case is a **sentinel value**, a value that when evaluated by the loop expression causes the loop to terminate.

The code `print(user_value*5)` produces a new string, which repeats the value of `user_value` 5 times. In this case, the value of `user_value` may be "-", thus the result of the multiplication is "----".

Another valid (but long and visually unappealing) method is the statement

```
print('{ }{}{}{}{}'.format(user_value, user_value, user_value, user_value, user_value))
```

Note that `input` may read in a multi-character string from the user, so only the first character is extracted from `user_input` with `user_value = user_input[0]`.

©zyBooks 09/27/22 11:37 469702

Once execution enters the loop body, execution continues to the body's end even if the expression becomes False midway through.

WGUC859v4

Figure 6.2.1: While loop example: Face-printing program that ends when user enters 'q'.

```
nose = '0' # Looks a little like a nose
user_value = '-'

while user_value != 'q':
    print(' {} {}'.format(user_value, user_value)) # Print eyes
    print(' {} '.format(nose)) # Print nose
    print(user_value*5) # Print mouth
    print('\n')

    # Get new character for eyes and mouth
    user_input = input("Enter a character ('q' for quit): \n")
    user_value = user_input[0]

print('Goodbye.\n')
```

```
- - -
0
-----
Enter a character ('q' for quit): x
x x
0
xxxxx

Enter a character ('q' for quit): @
@ @
0
@@@@@@

Enter a character ('q' for quit): q
Goodbye.
```

#### PARTICIPATION ACTIVITY

#### 6.2.3: Loop expressions.



Complete the loop expressions, using a single operator in your expression. Use the most straightforward translation of English to an expression.

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

- 1) Iterate while `x` is less than 100.



```
while [ ]:
    :
    # Loop body statements go here
```

**Check****Show answer**

- 2) Iterate while x is greater than or equal to 0.

**while**

:

```
# Loop body statements go  
here
```

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

**Check****Show answer**

- 3) Iterate while c equals 'g'.

**while**

:

```
# Loop body statements go  
here
```

**Check****Show answer**

- 4) Iterate *until* c equals 'z'.

**while**

:

```
# Loop body statements go  
here
```

**Check****Show answer**

## Stepping through a while loop

The following program animation provides another loop example. First, the user enters an integer. Then, the loop prints each digit one at a time starting from the right, using "% 10" to get the rightmost digit and "// 10" to remove that digit. The loop is only entered while num is greater than 0; once num reaches 0, the loop will have printed all digits.

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

**PARTICIPATION  
ACTIVITY**

6.2.4: While loop step-by-step.



### Animation content:

undefined

## Animation captions:

1. User enters the number 902. The first iteration prints "2".
2. The second iteration prints "0".
3. The third iteration prints "9", so every digit has been printed. The loop condition is checked one more time, and since num is 0, the loop stops.

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

## Example: While loop: Iterations

Each iteration of the program below prints one line with the year and the number of ancestors in that year. (Note: the program's output numbers are large due to not considering breeding among distant relatives, but nevertheless, a person has many ancestors.)

The program checks for `year_considered >= user_year` rather than for `year_considered != user_year`, because `year_considered` might be reduced past `user_year` without equaling it, causing an infinite loop. An **infinite loop** is a loop that will always execute because the loop's expression is always True. A common error is to accidentally create an infinite loop by assuming equality will be reached. Good practice is to include greater than or less than along with equality in a loop expression to help avoid infinite loops.

A program with an infinite loop may print output excessively, or just seem to stall (if the loop contains no printing). A user can halt a program by pressing Control-C in the command prompt running the Python program. Alternatively, some IDEs have a "Stop" button.

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

## zyDE 6.2.1: While loop example: Ancestors printing program.

Run the program below.

[Load default template](#)

```
1 year_considered = 2020 # Year being considered
2 num_ancestors = 2 # Approx. ancestors in considered year
3 years_per_generation = 20 # Approx. years per generation
4
5 user_year = int(input('Enter a past year (neg. for B.C.): '))
6 print()
7
8 while year_considered >= user_year:
9     print('Ancestors in {}: {}'.format(year_considered, num_ancestors))
10
11     num_ancestors = num_ancestors * 2
12     year_considered = year_considered - years_per_generation
13 |
```

1945

**Run**

**PARTICIPATION ACTIVITY**

6.2.5: While loop iterations.



What is the output of the following code? (Use "IL" for infinite loops.)

```
1) x = 0
   while x > 0:
       print(x, end=' ')
       x = x - 1
   print('Bye')
```

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4



**Check**

**Show answer**

2) `x = 5  
y = 18  
while y >= x:  
 print(y, end=' ')  
 y = y - x`

**Check****Show answer**

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

3) `x = 10  
while x != 3:  
 print(x, end=' ')  
 x = x / 2`

**Check****Show answer**

4) `x = 1  
y = 3  
z = 5  
while not (y < x < z):  
 print(x, end=' ')  
 x = x + 1`

**Check****Show answer**

**CHALLENGE ACTIVITY**

6.2.1: Enter the output of the while loop.

334598.939404.qx3zqy7

**Start**

Type the program's output

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

`g = 0  
while g <= 3:  
 print(g)  
 g += 1`

0  
1  
2  
3

1

2

3

4

**Check****Next****CHALLENGE ACTIVITY**

6.2.2: Basic while loop with user input.

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

Write an expression that executes the loop body as long as the user enters a non-negative number.

Note: If the submitted code has an infinite loop, the system will stop running the code after a few seconds and report "Program end never reached." The system doesn't print the test case that caused the reported message.

Sample outputs with inputs: 9 5 2 -1

**Body****Body****Body****Done.**

334598.939404.qx3zqy7

```
1 user_num = int(input())
2 while ''' Your solution goes here ''':
3     print('Body')
4     user_num = int(input())
5
6 print('Done.')
```

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4**Run**

View your last submission ▾

**CHALLENGE  
ACTIVITY****6.2.3: Basic while loop expression.**

Write a while loop that repeats while  $\text{user\_num} \geq 1$ . In each loop iteration, divide  $\text{user\_num}$  by 2, then print  $\text{user\_num}$ .

©zyBooks 09/27/22 11:37 469702

Steven Cameron  
WGUC859v4

Sample output with input: 20

```
10.0
5.0
2.5
1.25
0.625
```

Note: If the submitted code has an infinite loop, the system will stop running the code after a few seconds and report "Program end never reached." The system doesn't print the test case that caused the reported message.

334598.939404.qx3zqy7

```
1 user_num = int(input())
2
3 ''' Your solution goes here '''
4
```

©zyBooks 09/27/22 11:37 469702
Steven Cameron
WGUC859v4

**Run**

View your last submission ▾

## 6.3 More while examples

### Example: GCD

The following is an example of using a loop to compute a mathematical quantity. The program computes the greatest common divisor (GCD) among two user-entered integers `num_a` and `num_b`, using Euclid's algorithm: If `num_a > num_b`, set `num_a` to `num_a - num_b`, else set `num_b` to `num_b - num_a`. Repeat until `num_a` equals `num_b`, at which point `num_a` and `num_b` both equal the GCD.

zyDE 6.3.1: While loop example: GCD program.

Try running the program below that calculates the greatest common divisor (GCD) of positive integers.

```
Load default template...
20
15
Run
1 num_a = int(input('Enter first positive integer: '))
2 print()
3
4 num_b = int(input('Enter second positive integer: '))
5 print()
6
7 while num_a != num_b:
8     if num_a > num_b:
9         num_a = num_a - num_b
10    else:
11        num_b = num_b - num_a
12
13 print('GCD is {}'.format(num_a))
14
```

PARTICIPATION  
ACTIVITY

6.3.1: Loop example: Greatest common divisor.

©zyBooks 09/27/22 11:37 469/102  
Steven Cameron  
WGUC859v4

Use input values of `num_a = 15` and `num_b = 10` in the above GCD program. Try to answer the questions by mentally executing the statements. If stuck, consider adding additional print statements to the program.

- 1) What is the value of `num_a` before the first loop iteration?



**Check****Show answer**

- 2) What is num\_a after the first  
and before the second iteration?



©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

**Check****Show answer**

- 3) What is num\_b after the second  
and before the third iteration?

**Check****Show answer**

- 4) How many loop iterations will  
the algorithm execute?

**Check****Show answer**

---

## Example: Conversation

Below is a program that has a "conversation" with the user. The program asks the user to type something and then randomly prints one of four possible responses until the user enters "Goodbye". Note that the first few lines of the program represent a **docstring**: a multi-line string literal delimited at the beginning and end by triple-quotes. Either single ' or double " quotes can be used.

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

## zyDE 6.3.2: While loop example: Conversation.

Run the program below. Try adding additional conditions to leave the conversation, such as "See you later". Program input must end with the string 'Goodbye'.

[Load default template](#)  
©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

```
1 """
2 Program that has a conversation with the user.
3 Uses elif branching and a random number to mix up the program's responses
4 """
5 import random # Import a library to generate random numbers
6
7 print('Tell me something about yourself.')
8 print('You can type \'Goodbye\' at anytime to quit.\n')
9
10 user_text = input()
11
12 while user_text != 'Goodbye':
13     random_num = random.randint(0, 2) # Gives a random integer between 0 and 2
14     if random_num == 0:
15         print('\nPlease explain further.\n')
16     elif random_num == 1:
17         print("\nWhy do you say: '{}'?\n".format(user_text))
```

Thank you very much, Mr. Robot.  
Goodbye

**Run**

Each time through the while loop, the program will check if the user-entered string `user_text` is equal to the string literal "Goodbye". If the two strings are not equal, the while loop body executes. Within the while loop body, the program obtains a random number between 0 and 2 by using the random library. The `randint()` function provides a new random number each time the function is called. The arguments to `randint()`, 0 and 2, provide the minimum and maximum values that the function may return. Using the number given by `randint()`, the program's `elif` statements branch to a particular response.

**PARTICIPATION ACTIVITY**

6.3.2: Conversation program.



Refer to the above conversation program. If appropriate, type: unknown



- 1) Which if-else branch will execute if the user types "Goodbye"?  
Valid answers are branch 0, 1, 2 or none.

**Check****Show answer**

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4



- 2) How many times does the loop iterate in the program?

**Check****Show answer**

- 3) Write an expression using random.randint() that returns a number between 0 and 5, inclusive.

**Check****Show answer**

---

## Example: Getting input until a sentinel is seen

Loops are commonly used to process a series of input values. A sentinel value is used to terminate a loop's processing. The example below computes the average of an input list of positive integers, ending with 0. The 0 is not included in the average.

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

## zyDE 6.3.3: Computing average of a list with a sentinel.

[Load default template](#)

```
1 ...
2 Outputs average of list of positive integers
3 List ends with 0 (sentinel)
4 Ex: 10 1 6 3 0  yields (10 + 1 + 6 + 3) / 4, or 5
5 ...
6
7 values_sum = 0
8 num_values = 0
9
10 curr_value = int(input())
11
12 while curr_value > 0: # Get values until 0 (or less)
13     values_sum += curr_value
14     num_values += 1
15     curr_value = int(input())
16
17 print('Average: {:.0f}\n'.format(values_sum / num_values))
```

```
10
1
6
```

**Run****PARTICIPATION ACTIVITY**

6.3.3: Average example with a sentinel.



Consider the example above and the given example input sequence 10 1 6 3 0.

- 1) How many actual (non-sentinel) values are given in the first input sequence?

- 1
- 4
- 5



©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

- 2) For the given input sequence, what is the final value of num\_values?



0 4

- 3) Suppose the first input was 0. Would  
values\_sum / num\_values be 0?

 Yes No

- 4) What would happen if the following  
list was input: 10 1 6 3 -1?

 Output would be 5 Output would be 4 Error

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

**CHALLENGE  
ACTIVITY****6.3.1: While loop with sentinel.**

334598.939404.qx3zqy7

**Start**

Type the program's output

```
entered_value = int(input())  
  
max_value = entered_value  
  
while entered_value > 0:  
    if entered_value > max_value:  
        max_value = entered_value  
    entered_value = int(input())  
  
print('Max value:', max_value, end='')
```

**Input**

2  
34  
12  
0

**Output**

Max value: 34

**1**

©zyBooks 09/27/22 11:37 469702

Steven Cameron  
WGUC859v4

**Check****Next****CHALLENGE  
ACTIVITY****6.3.2: Bidding example.**

Write an expression that continues to bid until the user enters 'n'.

Sample output with inputs: 'y' 'y' 'n'

```
I'll bid $7!
Continue bidding? I'll bid $15!
Continue bidding? I'll bid $23!
Continue bidding?
```

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

334598.939404.qx3zqy7

```
1 import random
2 random.seed(5)
3
4 keep_bidding = '-'
5 next_bid = 0
6
7 while ''' Your solution goes here ''':
8     next_bid = next_bid + random.randint(1, 10)
9     print('I\'ll bid ${}'.format(next_bid))
10    print('Continue bidding?', end=' ')
11    keep_bidding = input()
```

Run

View your last submission ▾

CHALLENGE ACTIVITY

6.3.3: While loop: Insect growth.



Given positive integer num\_insects, write a while loop that prints, then doubles, num\_insects each iteration. Print values  $\leq 100$ . Follow each number with a space.

Sample output with input: 8

8 16 32 64

334598.939404.qx3zqy7

```
1 num_insects = int(input()) # Must be >= 1
2
3 ''' Your solution goes here '''
4
```

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

**Run**

View your last submission ▾

## 6.4 Counting

### Counting with a while loop

Commonly, a loop should iterate a specific number of times, such as 10 times. The programmer can use a variable to count the number of iterations, called a **loop variable**. To iterate N times using an integer loop variable i, a loop<sup>1</sup> with the following form is used:

Construct 6.4.1: Counting while loop form.

```
# Iterating N times using a loop
variable
i = 1
while i <= N:
    # Loop body statements go here
    i = i + 1
```

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

A common error is to forget to include the loop variable update (e.g.,  $i = i + 1$ ) at the end of the loop, causing an unintended infinite loop.

The following program outputs the amount of money in a savings account each year for the user-entered number of years, with \$10,000 initial savings and 5% yearly interest:

### zyDE 6.4.1: While loop that counts iterations: Savings interest program.

[Load default template](#)

©zyBooks 09/27/22 11:37 469702

Steven Cameron

```
1 '''Program that calculates savings and interest''' WGUC859v4
2
3 initial_savings = 10000
4 interest_rate = 0.05
5
6 print('Initial savings of ${}'.format(initial_savings))
7 print('at {:.0f}% yearly interest.\n'.format(interest_rate*100))
8
9 years = int(input('Enter years: '))
10 print()
11
12 savings = initial_savings
13 i = 1 # Loop variable
14 while i <= years: # Loop condition
15     print(' Savings at beginning of year {}: ${:.2f}'.format(i, savings))
16     savings = savings + (savings*interest_rate)
17     i = i + 1 # Increment loop variable
```

10

**Run**

PARTICIPATION ACTIVITY

6.4.1: Savings interest program.



Refer to the program above.

- 1) With an initial savings of \$10000 and interest rate of 0.05, what's the amount of savings at the beginning of year 4? Ignore cents and do not include the dollar sign (\$).

©zyBooks 09/27/22 11:37 469702

Steven Cameron

WGUC859v4

**Check****Show answer**

- 2) If interest\_rate is 3% and initial\_savings are \$5000, savings will be greater than \$7500 after how many loop iterations?



©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

**Check****Show answer****PARTICIPATION ACTIVITY**

6.4.2: Basic counting with while loops.



Use <= in each answer.

- 1) Loop iterates 10 times.

```
i = 1
while [REDACTED]
:
# Loop body statements go
here
i = i + 1
```

**Check****Show answer**

- 2) Loop iterates 99 times.

```
i = 1
while [REDACTED]
:
# Loop body statements go
here
i = i + 1
```



©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

**Check****Show answer**

- 3) Loop iterates 2 times.



```
i = 1
while [REDACTED]
:
    # Loop body statements go
here
    i = i + 1
```

## Other forms of counting

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

Counting down is also common, as in counting down from 5 to 1. The loop body executes when i is 5, 4, 3, 2, and 1, but does not execute when i reaches 0.

Figure 6.4.1: While loop with loop variable that counts down.

```
i = 5
while i >= 1:
    # Loop body statements go
here
    i = i - 1
```

The loop body executes when i is 5, 4, 3, 2, and 1, but does not execute when i reaches 0.

Counting sometimes occurs by steps greater than 1. Ex: A loop that prints even values from 0 to 100 (i.e., counts from 0 to 100 by 2s) is:

Figure 6.4.2: Loop variable increased by 2 per iteration.

```
i = 0
while i <= 100:
    # Loop body statements go
here
    i = i + 2
```

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

## zyDE 6.4.2: Loop over presidential election years.

Write a program that prints the U.S. presidential election years from 1792 to present knowing that such elections occur every 4 years.

Hint: Initialize your loop variable to 1792. Don't forget to use `<=` rather than `==` to help infinite loop.

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

[Load default template...](#)

**Run**

```
1 year = 1792
2 current_year = ?
3
4 while year ? ??:
5     # Print the election year
6     year = year + ?
7 |
```

### PARTICIPATION ACTIVITY

#### 6.4.3: Forms of counting.



Complete the missing parts of the code.

- 1) Loop iterates over the odd integers from 1 to 9 (inclusive).



```
i = 1
while i <= 9:
    # Loop body statements go here
```

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

**Check**

**Show answer**

- 2) Loop iterates over multiples of 5 from 0 to 1000 (inclusive).



```
i = 0
while :
    # Loop body statements go
here
    i = i + 5
```

**Check****Show answer**

- 3) Loop iterates over the odd integers from 211 down to 31 (inclusive).

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4



```
i = 211
while i >= 31:
    # Loop body statements go
here
```

**Check****Show answer**

- 4) Loop iterates from -100 to 65.



```
while i <= 65:
    # Loop body statements go
here
    i = i + 1
```

**Check****Show answer****PARTICIPATION ACTIVITY**

6.4.4: Counting in a loop simulator.



The following tool allows you to enter values for a loop's parts, and then executes the loop. Using the tool, try to solve each listed problem individually.

©zyBooks 09/27/22 11:37 469702

The tool can use any relational or equality operator, such as `<`, `<=`, `>`, `>=`, `==`, etc. Identity and membership operators like "is" or "in" will not work.

Steven Cameron  
WGUC859v4

1. 0 to 10,000 by 500s (0, 500, 1000, ...)
2. -19 to 19 by 1s
3. 10 to -10 by 1s
4. Multiples of 3 between 0 and 100
5. Powers of 2 from 1 to 256 (1, 2, 4, 8, ...)

## 6. Come up with your own challenges

```
i =   
while i   :   
    print(i, end=' ')  
    i = i    
  
Run code
```

Output is:

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

### zyDE 6.4.3: Calculate a factorial.

Write a program that lets a user enter N and that outputs N! (N factorial, meaning  $N * (N-1) * (N-2) * \dots * 2 * 1$ ). Hint: Use a loop variable i that counts from total-1 down to 1. Compare output with some of these answers: 1:1, 2:2, 3:6, 4:24, 5:120, 8:40320.

Load default template...

```
1 N = int(input()) # Read user-entered number  
2 total = N  
3 # Initialize the Loop variable  
4  
5 while i ? ?:  
6     # Set total to total * (i)  
7     # Decrement i  
8  
9 print(total)  
10
```

5

Run

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

## Shorthand operators

Because assignments such as `i = i + 1` are so common in programs, the programming language provides a shorthand version: `i += 1`. Similar operators include `+=`, `-=`, `*=`, and `/=`. For example, `num *= x` is shorthand for `num = num * x`. The item on the right can be an expression, so `num *= x + y` is shorthand for `num = num * (x + y)`. Usage of such operators is common in loops.

## Construct 6.4.2: Operators like `+=` are common in loops.

```
i = 0
while i < N:
    # Loop body statements go
    here
    i += 1
```

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

### PARTICIPATION ACTIVITY

#### 6.4.5: Shorthand operators.



Answer each question using the operators of the form `+=`, `*=`, `/=`, `-=`, etc.

- 1) Write a statement using `*=` that doubles the value of a variable `my_var`.

**Check****Show answer**

- 2) Write a statement using `+=` that is equivalent to

```
my_var = my_var + my_var
/ 2
```

**Check****Show answer**

### CHALLENGE ACTIVITY

#### 6.4.1: Loops with variables that count.



334598.939404.qx3zqy7

**Start**

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

Type the program's output

```
n = 1
while n < 4:
    print(n - 2)
    n = n + 1
```

-1  
0  
1

1

2

©zyBooks 09/27/23 11:37 469702
Steven Cameron
WGUC859v4

**CHALLENGE**  
**ACTIVITY**

6.4.2: While loop: Print 1 to N.



Write a while loop that prints from 1 to user\_num, increasing by 1 each time.

Sample output with input: 4

1  
2  
3  
4

334598.939404.qx3zqy7

```
1 i = 1
2
3 user_num = int(input()) # Assume positive
4
5 ''' Your solution goes here '''
6
```

©zyBooks 09/27/22 11:37 469702
Steven Cameron
WGUC859v4

Run

View your last submission ▾

**CHALLENGE  
ACTIVITY**

## 6.4.3: Printing output using a counter.



Retype or copy, and then run the following code; note incorrect behavior. Then fix errors in the code, which should print num\_stars asterisks.

```
while num_printed != num_stars:  
    print('*')
```

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

Sample output with input: 3

```
*  
*  
*
```

334598.939404.qx3zqy7

```
1 num_printed = 0  
2  
3 num_stars = int(input())  
4  
5 ''' Your solution goes here '''  
6
```

**Run**

View your last submission ▾

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

(\*1) Focus is placed on mastering basic looping using while loops, before introducing for loops and range().

## 6.5 For loops

## Survey

The following questions are part of a zyBooks survey to help us improve our content so we can offer the best experience for students. The survey can be taken anonymously. Please take a short moment to answer below.

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

Link: [Student survey](#)

## Basics

A common programming task is to access all of the elements in a container. Ex: Printing every item in a list. A **for loop** statement loops over each element in a container one at a time, assigning a variable with the next element that can then be used in the loop body. The container in the for loop statement is typically a list, tuple, or string. Each iteration of the loop assigns the name given in the for loop statement with the next element in the container.

### Construct 6.5.1

```
for variable in container:  
    # Loop body: Sub-statements to execute  
    # for each item in the container  
  
    # Statements to execute after the for loop is  
    # complete
```

PARTICIPATION  
ACTIVITY

6.5.1: Iterating over a list using a for loop.



### Animation content:

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

undefined

### Animation captions:

1. The first iteration assigns the variable name with 'Bill' and prints 'Hi Bill!' to the screen.
2. The second iteration assigns the variable name with 'Nicole' and prints 'Hi Nicole!'.
3. The third iteration assigns the variable name with 'John' and prints 'Hi John!'.

The for loop above iterates over the list `['Bill', 'Nicole', 'John']`. The first iteration assigns the variable name with 'Bill', the second iteration assigns name with 'Nicole', and the final iteration assigns name with 'John'. For sequence types like lists and tuples, the assignment order follows the position of the elements in the container, starting with position 0 (the leftmost element) and continuing until the last element is reached.

Iterating over a dictionary using a for loop assigns the loop variable with the keys of the dictionary. The values can then be accessed using the key.

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

Figure 6.5.1: A for loop assigns the loop variable with a dictionary's keys.

```
channels = {  
    'MTV': 35,  
    'CNN': 28,  
    'FOX': 11,  
    'NBC': 4,  
    'CBS': 12  
}  
  
for c in channels:  
    print('{} is on channel {}'.format(c,  
                                         channels[c]))
```

MTV is on channel  
35  
CNN is on channel  
28  
FOX is on channel  
11  
NBC is on channel 4  
CBS is on channel  
12

A for loop can also iterate over a string. Each iteration assigns the loop variable with the next character of the string. Strings are sequence types just like lists, so the behavior is identical (leftmost character first, then each following character).

Figure 6.5.2: Using a for loop to access each character of a string.

```
my_str = ''  
for character in "Take me to the  
moon.":  
    my_str += character + '_'  
print(my_str)
```

T\_a\_k\_e\_ \_m\_e\_ \_t\_o\_ \_t\_h\_e\_  
\_m\_o\_o\_n\_.\_

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

PARTICIPATION  
ACTIVITY

6.5.2: Creating for loops.



Complete the for loop statement by giving the loop variable and container.



- 1) Iterate over the given list using a variable called my\_pet.

```
for  in ['Scooter',  
    'Kobe', 'Bella']:  
    # Loop body statements
```

**Check****Show answer**

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4



- 2) Iterate over the list my\_prices using a variable called price.

```
for  :  
    # Loop body statements
```

**Check****Show answer**

- 3) Iterate the string '911' using a variable called number.

```
for  :  
    # Loop body statements
```

**Check****Show answer**

## For loop examples

For loops can be used to perform some action during each loop iteration. A simple example would be printing the value, as above examples demonstrated. The program below uses an additional variable to sum list elements to calculate weekly revenue and an average daily revenue.

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

Figure 6.5.3: For loop example: Calculating shop revenue.

```
daily_revenues = [
    2356.23, # Monday
    1800.12, # Tuesday
    1792.50, # Wednesday
    2058.10, # Thursday
    1988.00, # Friday
    2002.99, # Saturday
    1890.75 # Sunday
]

total = 0
for day in daily_revenues:
    total += day

average = total / len(daily_revenues)

print('Weekly revenue: ${:.2f}'.format(total))
print('Daily average revenue:
${:.2f}'.format(average))
```

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

Weekly revenue: \$13888.69  
Daily average revenue:  
\$1984.10

A for loop may also iterate backwards over a sequence, starting at the last element and ending with the first element, by using the **reversed()** function to reverse the order of the elements.

Figure 6.5.4: For loop example: Looping over a sequence in reverse.

The following program first prints a list that is ordered alphabetically, then prints the same list in reverse order.

```
names = [
    'Biffle',
    'Bowyer',
    'Busch',
    'Gordon',
    'Patrick'
]

for name in names:
    print(name, '|', end=' ')

print('\nPrinting in reverse:')
for name in reversed(names):
    print(name, '|', end=' ')
```

Biffle | Bowyer | Busch | Gordon | Patrick |
Printing in reverse:
Patrick | Gordon | Busch | Bowyer | Biffle |

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

**PARTICIPATION  
ACTIVITY**

## 6.5.3: For loops.



Fill in the missing code to perform the desired calculation.

- 1) Compute the average number of kids.



```
# Each list item is the
# number of kids in a family.
num_kids = [1, 1, 2, 2, 1, 4,
3, 1]
```

```
total = 0
for num in num_kids:
    total +=
```

```
average = total /
len(num_kids)
```

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

**Check****Show answer**

- 2) Assign num\_neg with the number of below-freezing Celsius temperatures in the list.



```
temperatures = [30, 20, 2,
-5, -15, -8, -1, 0, 5, 35]
```

```
num_neg = 0
for temp in temperatures:
    if temp < 0:
```

**Check****Show answer**

- 3) Print scores in order from highest to lowest. Note: List is pre-sorted from lowest to highest.

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

```
scores = [75, 77, 80, 85, 90,  
95, 99]
```

```
for scr in
```

```
.
```

**CHALLENGE  
ACTIVITY**

6.5.1: Looping over strings, lists, and dictionaries.

©zyBooks 09/27/22 11:37 469702

Steven Cameron  
WGUC859v4

334598.939404.qx3zqy7

Start

Type the program's output

```
colors = ['blue', 'gold', 'pink']  
for color in colors:  
    print(color)
```

blue  
gold  
pink

1

2

3

4

Check

Next

**CHALLENGE  
ACTIVITY**

6.5.2: For loop: Printing a list



Write an expression to print each price in stock\_prices.

Sample output with inputs: 34.62 76.30 85.05

```
$ 34.62  
$ 76.30  
$ 85.05
```

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

334598.939404.qx3zqy7

```
1 # NOTE: The following statement converts the input into a list container  
2 stock_prices = input().split()  
3  
4 for ''' Your solution goes here ''':  
5     print('$'. price)
```

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

Run

View your last submission ▾

CHALLENGE  
ACTIVITY

6.5.3: For loop: Printing a dictionary



Write a for loop to print each contact in contact\_emails.

Sample output with inputs: 'Alf' 'alf1@hmail.com'

```
mike.filt@bmail.com is Mike Filt
s.reyn@email.com is Sue Reyn
narty042@nmail.com is Nate Arty
alf1@hmail.com is Alf
```

334598.939404.qx3zqy7

```
1 contact_emails = {
2     'Sue Reyn' : 's.reyn@email.com',
3     'Mike Filt': 'mike.filt@bmail.com',
4     'Nate Arty': 'narty042@nmail.com'
5 }
6
7 new_contact = input()
8 new_email = input()
9 contact_emails[new_contact] = new_email
10
11 ''' Your solution goes here '''
12
```

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

**Run**

View your last submission ▾

## 6.6 Counting using the range() function

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

### The range() function

While loops are commonly used for counting a specific number of iterations, and for loops are commonly used to iterate over all elements of a container. The range() function allows counting in for loops as well. **range()** generates a sequence of integers between a starting integer that is included in the range, an ending integer that is not included in the range, and an integer step value. The sequence is generated by starting at the start integer and incrementing by the step value until the ending integer is reached or surpassed.

The range() function can take up to three integer arguments.

- **range(Y)** generates a sequence of all non-negative integers less than Y.  
Ex: **range(3)** creates the sequence 0, 1, 2.
- **range(X, Y)** generates a sequence of all integers  $\geq X$  and  $< Y$ .  
Ex: **range(-7, -3)** creates the sequence -7, -6, -5, -4.
- **range(X, Y, Z)**, where Z is positive, generates a sequence of all integers  $\geq X$  and  $< Y$ , incrementing by Z.  
Ex: **range(0, 50, 10)** creates the sequence 0, 10, 20, 30, 40.
- **range(X, Y, Z)**, where Z is negative, generates a sequence of all integers  $\leq X$  and  $> Y$ , incrementing by Z.  
Ex: **range(3, -1, -1)** creates the sequence 3, 2, 1, 0.

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

Table 6.6.1: Using the range() function.

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

Evaluating the range() function creates a new "range" type object. Ranges represent an arithmetic progression, i.e., some sequence of integers with a start, end, and step between integers. The range type is a sequence type like lists and tuples, but is immutable. In general, range objects are only used as a part of a for loop statement.

## zyDE 6.6.1: For loop example: Calculating yearly savings.

The below program uses a for loop to calculate savings and interest. Try changing the function to print every three years instead, using the three-argument alternate version range(). Modify the interest calculation inside the loop to compute three years worth savings instead of one.

[Load default template...](#)

©zyBooks 09/27/22 11:37 469702

Steven Cameron  
WGUC859v4

8

**Run**

```
1 '''Program that calculates savings and
2
3 initial_savings = 10000
4 interest_rate = 0.05
5
6 years = int(input('Enter years: '))
7 print()
8
9 savings = initial_savings
10 for i in range(years):
11     print(' Savings in year {}: ${:.2f}'.format(i, savings))
12     savings = savings + (savings*interest_rate)
13
14 print('\n')
15 |
```

### PARTICIPATION ACTIVITY

6.6.1: The range() function.



1) What sequence is generated by

`range(7)`?



- 0 1 2 3 4 5 6 7
- 1 2 3 4 5 6
- 0 1 2 3 4 5 6

2) What sequence is generated by

`range(2, 5)`?



- 2 3 4
- 2 3 4 5
- 0 1 2 3 4

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

**PARTICIPATION  
ACTIVITY**

## 6.6.2: The range() function.



Write the simplest range() function that generates the appropriate sequence of integers.

- 1) Every integer from 0 to 500.

**Check****Show answer**

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

- 2) Every integer from 10 to 20

**Check****Show answer**

- 3) Every 2nd integer from 10 to 20

**Check****Show answer**

- 4) Every integer from 5 down to -5

**Check****Show answer****CHALLENGE  
ACTIVITY**

## 6.6.1: Enter the for loop's output.

334598.939404.qx3zqy7

**Start**

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

```
for i in range(4):  
    print(i, end=' ')
```

0 1 2 3

1

2

[Check](#)[Next](#)

## 6.7 While vs. for loops

©zyBooks 09/27/22 11:37 469702

Steven Cameron  
WGUC859v4

### While loop and for loop correspondence

Both for loops and while loops can be used to count a specific number of loop iterations. A for loop combined with range() is generally preferred over while loops, since for loops are less likely to become stuck in an infinite loop situation. A programmer may easily forget to increment a while loop's variable (causing an infinite loop), but for loops iterate over a finite number of elements in a container and are thus guaranteed to complete.

**PARTICIPATION ACTIVITY**

6.7.1: While/for loop correspondence.



#### Animation captions:

1. A for loop and range function can replace a while loop.
2. A more concise for loop uses a single argument for range().

As a general rule:

1. Use a *for loop* when the number of iterations is computable before entering the loop, as when counting down from X to 0, printing a string N times, etc.
2. Use a *for loop* when accessing the elements of a container, as when adding 1 to every element in a list, or printing the key of every entry in a dict, etc.
3. Use a *while loop* when the number of iterations is not computable before entering the loop, as when iterating until a user enters a particular character.

These are not hard rules, just general guidelines.

**PARTICIPATION ACTIVITY**

6.7.2: While loops and for loops.

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

Indicate whether a while loop or for loop should be used in the following scenarios:

- 1) Iterate as long as the user-entered string c is not q.





- 2) Iterate until the values of x and y are equal, where x and y are changed in the loop body.

- while
- for

- 3) Iterate 1500 times

- while
- for

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4



## 6.8 Nested loops

### Nested loops

A **nested loop** is a loop that appears as part of the body of another loop. The nested loops are commonly referred to as the **outer loop** and **inner loop**.

Nested loops have various uses. One use is to generate all combinations of some items. Ex: The following program generates all two letter .com Internet domain names. Recall that `ord()` converts a 1-character string into an integer, and `chr()` converts an integer into a character. Thus, `chr(ord('a') + 1)` results in 'b'.

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

Figure 6.8.1: Nested loops example: Two-letter domain name printing program.

```
"""
Program to print all 2-letter domain names.

Note that ord() and chr() convert between text and the ASCII or
Unicode encoding:
- ord('a') yields the encoded value of 'a', the number 97.
- ord('a')+1 adds 1 to the encoded value of 'a', giving 98.
- chr(ord('a')+1) converts 98 back into a letter, producing 'b'
"""

print('Two-letter domain names:')

letter1 = 'a'
letter2 = '?'
while letter1 <= 'z': # Outer loop
    letter2 = 'a'
    while letter2 <= 'z': # Inner loop
        print('{}{}.com'.format(letter1, letter2))
        letter2 = chr(ord(letter2) + 1)
    letter1 = chr(ord(letter1) + 1)
```

#### Two-letter domain

names:

©zyBooks 09/27/22 11:37 469702  
aa.com  
ab.com  
ac.com  
ad.com  
ae.com  
af.com  
ag.com  
ah.com  
ai.com  
aj.com  
ak.com  
al.com  
am.com  
an.com  
ao.com  
ap.com  
aq.com  
ar.com  
as.com  
at.com  
au.com  
av.com  
aw.com  
ax.com  
ay.com  
az.com  
ba.com  
bb.com  
...  
zy.com  
zz.com

(Forget about buying a two-letter domain name: They are all taken, and each sells for several hundred thousand or millions of dollars. Source: dnjournal.com, 2012.)

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

## zyDE 6.8.1: Two character dotcom domain names.

Modify the program to include two-character .com names, where the second character is a letter or a number, e.g., a2.com. Hint: Add a second while loop nested in the outer loop, following the first inner loop, that iterates through the numbers 0-9.

[Load default template...](#)

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

```
1 """
2 Program to print all 2-letter domain
3 Note that ord() and chr() convert between
4 ord('a') is 97. ord('b') is 98, and
5 """
6 print('Two-letter domain names:')
7
8 letter1 = 'a'
9 letter2 = '?'
10 while letter1 <= 'z': # Outer Loop
11     letter2 = 'a'
12     while letter2 <= 'z': # Inner Loop
13         print('{}{}.com'.format(letter1, letter2))
14         letter2 = chr(ord(letter2) + 1)
15     letter1 = chr(ord(letter1) + 1)
16
17 |
```

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

## zyDE 6.8.2: Nested loop example: Histogram.

Here is a nested loop example that graphically depicts an integer's magnitude by using asterisks, creating what is commonly called a *histogram*:

Run the program below and observe the output. Modify the program to print one asterisk per unit. So if the user enters 40, print 8 asterisks.

©zyBooks 09/27/22 11:37 469702  
Steven Cameron

WGUC859v4  
5  
10  
-1

Run

```
Load default template...

1 num = 0
2 while num >= 0:
3     num = int(input('Enter an integer
4
5     if num >= 0:
6         print('Depicted graphically:')
7         for i in range(num):
8             print('*', end=' ')
9         print('\n')
10
11 print('Goodbye.')
12 |
```

PARTICIPATION  
ACTIVITY

### 6.8.1: Nested loops.



- 1) Given the following code, how many times will the inner loop body execute?

```
for i in range(2):
    for j in range(3):
        # Inner loop body
```



**Check**

**Show answer**

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

- 2) Given the following code, how many times will the print statement execute?



```
for i in range(5):
    for j in range(10, 12):
        t(i,
```

**Check****Show answer**

- 3) What is the output of the following code?

```
c1 = 'a'
while c1 < 'b':
    c2 = 'a'
    while c2 <= 'c':
        print('{}{}'.format(c1,
c2), end=' ')
        c2 = chr(ord(c2) + 1)
    c1 = chr(ord(c1) + 1)
```

**Check****Show answer**

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

- 4) What is the output of the following code?

```
i1 = 1
while i1 < 19:
    i2 = 3
    while i2 <= 9:
        print('{}
{}'.format(i1,i2), end=' ')
        i2 = i2 + 3
    i1 = i1 + 10
```

**Check****Show answer****CHALLENGE ACTIVITY****6.8.1: Nested loops: Print rectangle**

Given the number of rows and the number of columns, write nested loops to print a rectangle.

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

Sample output with inputs: 2 3

```
* * *
* * *
```

334598.939404.qx3zqy7

```
1 num_rows = int(input())
2 num_cols = int(input())
3
4 ''' Your solution goes here '''
5
6     print('*', end=' ')
7 print()
```

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

**Run**

View your last submission ▾

**CHALLENGE ACTIVITY**

6.8.2: Nested loops: Print seats.



Given num\_rows and num\_cols, print a list of all seats in a theater. Rows are numbered, columns lettered, as in 1A or 3E. Print a space after each seat.

Sample output with inputs: 2 3

1A 1B 1C 2A 2B 2C

334598.939404.qx3zqy7

```
1 num_rows = int(input())
2 num_cols = int(input())
3
4 # Note 1: You will need to declare more variables
5 # Note 2: Place end=' ' at the end of your print statement to separate seats by space
6
7 ''' Your solution goes here '''
8
9 print()
```

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

[Run](#)

View your last submission ▾

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

## 6.9 Developing programs incrementally

### Incremental programming

As programs increase in complexity, a programmer's development process becomes more important. A programmer should not write the entire program and then run the program—hoping the program works. If, as is often the case, the program does not work on the first try, debugging at that point can be extra difficult because the program may have many distinct bugs.

Experienced programmers practice **incremental programming**, starting with a simple version of the program, and then growing the program little-by-little into a complete version.



### Example: Phone number program

The following program allows the user to enter a phone number that includes letters, which appear on phone keypads along with numbers and are commonly used by companies as a marketing tactic (e.g., 1-555-HOLIDAY). The program then outputs the phone number using numbers only.

The first program version simply prints each element of the string to ensure the loop iterates properly through each string element.

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

Figure 6.9.1: First version echoes input phone number string.

```
user_input = input('Enter phone number: ')  
  
index = 0  
for character in user_input:  
    print('Element {} is: {}'.format(index,  
character))  
    index += 1
```

```
Enter phone number: 1-555-  
HOLIDAY  
Element 0 is: 1  
Element 1 is: -  
Element 2 is: 5  
Element 3 is: 5  
Element 4 is: 5  
Element 5 is: -  
Element 6 is: H  
Element 7 is: O  
Element 8 is: L  
Element 9 is: I  
Element 10 is: D  
Element 11 is: A  
Element 12 is: Y
```

The second program version outputs the numbers (0 - 9) of the phone number and outputs a '?' for all other characters. A **FIXME comment** attracts attention to code that needs to be fixed in the future. Many editors automatically highlight FIXME comments. Large projects with multiple programmers might also include a username and date, as in **FIXME(01/22/2018, John)**.

Figure 6.9.2: Second version echoes numbers, and has FIXME comment.

```
user_input = input('Enter phone number: ')  
phone_number = ''  
  
for character in user_input:  
    if '0' <= character <= '9':  
        phone_number += character  
    else:  
        #FIXME: Add elif branches for letters and  
hyphen  
        phone_number += '?'  
  
print('Numbers only: {}'.format(phone_number))
```

```
Enter phone number: 1-555-  
HOLIDAY  
Numbers only: 1?555???????
```

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

The third version completes the elif branch for the letters A-C (lowercase and uppercase, per a standard phone keypad). The code also modifies the if branch to echo a hyphen in addition to numbers. The third version adds the elif branch for the letters A-C (lowercase and uppercase, per a standard phone keypad).

Figure 6.9.3: Third version echoes hyphens too, and handles first three letters.

```
user_input = input('Enter phone number: ')
phone_number = ''

for character in user_input:
    if ('0' <= character <= '9') or (character == '-'):
        phone_number += character
    elif ('a' <= character <= 'c') or ('A' <= character
<= 'C'):
        phone_number += '2'
    #FIXME: Add remaining elif branches
    else:
        phone_number += '?'

print('Numbers only: {}'.format(phone_number))
```

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

```
Enter phone number: 1-555-
HOLIDAY
Numbers only: 1-555-?????2?
```

The fourth version can be created by filling in the elif branches similarly for other letters and adding more instructions for handling unexpected characters. The code is not shown below, but sample input/output is provided.

Figure 6.9.4: Fourth and final version sample input/output.

```
Enter phone number (letters/- OK, no spaces): 1-555-HOLIDAY
Numbers only: 1-555-4654329
...
Enter phone number (letters/- OK, no spaces): 1-555-holiday
Numbers only: 1-555-4654329
...
Enter phone number (letters/- OK, no spaces): 999-9999
Numbers only: 999-9999
...
Enter phone number (letters/- OK, no spaces): 9876zywx%$#@
Numbers only: 98769999????
```

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

## zyDE 6.9.1: Complete the phone number program.

Complete the program by providing the additional branches for decoding other letters in the phone number. Try incrementally writing the program by adding one elif branch at a time and testing that each added branch works as intended.

[Load default template...](#)

```
1 user_input = input('Enter phone number')
2 phone_number = ''
3
4 for character in user_input:
5     if ('0' <= character <= '9') or (character >= 'A' and character <= 'Z'):
6         phone_number += character
7     elif ('a' <= character <= 'c') or (character >= 'a' and character <= 'z'):
8         phone_number += '2'
9     #FIXME: Add remaining elif branches
10    else:
11        phone_number += '?'
12
13 print('Numbers only: {}'.format(phone_number))
```

1-555-HOLIDAY 22 11:37 469702

Steven Cameron  
WGUC859v4

[Run](#)

### PARTICIPATION ACTIVITY

#### 6.9.1: Incremental programming.



- 1) Incremental programming may help reduce the number of errors in a program.

True  
 False



- 2) FIXME comments provide a way for a programmer to remember what needs to be added.

True  
 False



- 3) Once a program is complete, one would expect to see several FIXME comments.



©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

True

False

## 6.10 Break and continue

©zyBooks 09/27/22 11:37 469702

Steven Cameron

WGUC859v4

### Break statements

A **break** statement in a loop causes the loop to exit immediately. A break statement can sometimes yield a loop that is easier to understand.

In the example below, the nested for loops generate possible meal options for the number of empanadas and tacos that can be purchased. The inner loop body calculates the cost of the current meal option. If the meal cost is equal to the user's amount of money, the search is over, so the break statement immediately exits the inner loop. The outer loop body also checks if the meal cost and the user's amount of money are equal, and if so, that break statement exits the outer loop.

The program could be written without break statements, but the loops' condition expressions would be more complex and the program would require additional code, perhaps being harder to understand.

©zyBooks 09/27/22 11:37 469702

Steven Cameron

WGUC859v4

Figure 6.10.1: Break statement.

```
empanada_cost = 3
taco_cost = 4

user_money = int(input('Enter money for meal: '))
max_empanadas = user_money // empanada_cost
max_tacos = user_money // taco_cost

meal_cost = 0
for num_tacos in range(max_tacos + 1):
    for num_empanadas in range(max_empanadas + 1):
        meal_cost = (num_empanadas * empanada_cost) + (num_tacos *
taco_cost)

    # Find first meal option that exactly matches user money
    if meal_cost == user_money:
        break

    # Find first meal option that exactly matches user money
    if meal_cost == user_money:
        break

if meal_cost == user_money:
    print('${} buys {} empanadas and {} tacos without change.'
          .format(meal_cost, num_empanadas, num_tacos))
else:
    print('You cannot buy a meal without having change left over.') 
```

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

```
Enter money for meal: 20
$20 buys 4 empanadas and 2 tacos without change.
...
Enter money for meal: 31
$31 buys 9 empanadas and 1 tacos without change.
```

**PARTICIPATION ACTIVITY****6.10.1: Break statements.**

Given the following while loop, what is the value variable z is assigned with for the given values of variables a, b and c?

```
mult = 0
while a < 10:
    mult = b * a
    if mult > c:
        break
    a = a + 1
z = a
```

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

- 1) a = 1, b = 1, c = 0



**Check****Show answer**2)  $a = 4, b = 5, c = 20$ **Check****Show answer**

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

## Continue statements

A **continue** statement in a loop causes an immediate jump to the while or for loop header statement. A continue statement can improve the readability of a loop. The example below extends the previous meal finder program to find meal options for which the total number of items purchased is evenly divisible by the number of diners. In addition, the following program will output all possible meal options, instead of reporting the first meal option found.

The program uses two nested for loops to try all possible combinations of tacos and empanadas. If the total number of tacos and empanadas is not exactly divisible by the number of diners (e.g., `num_tacos + num_empanadas % num_diners != 0`), the continue statement will immediately proceed to the next iteration of the for loop.

Break and continue statements can be helpful to avoid excessive indenting/nesting within a loop. However, because someone reading a program could easily overlook a break or continue statement, such statements should be used only when their use is clear to the reader.

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

Figure 6.10.2: Continue statement.

```
empanada_cost = 3
taco_cost = 4

user_money = int(input('Enter money for meal: '))
num_diners = int(input('How many people are eating: '))

max_empanadas = user_money // empanada_cost
max_tacos = user_money // taco_cost

meal_cost = 0
num_options = 0
for num_tacos in range(max_tacos + 1):
    for num_empanadas in range(max_empanadas + 1):

        # Total items purchased must be equally divisible by number of
        # diners
        if (num_tacos + num_empanadas) % num_diners != 0:
            continue

        meal_cost = (num_empanadas * empanada_cost) + (num_tacos * taco_cost)

        if meal_cost == user_money:
            print('${} buys {} empanadas and {} tacos without change.'
                  .format(meal_cost, num_empanadas, num_tacos))
            num_options += 1

if num_options == 0:
    print('You cannot buy a meal without having change left over.') 
```

```
Enter money for meal: 60
How many people are eating: 3
$60 buys 12 empanadas and 6 tacos without change.
$60 buys 0 empanadas and 15 tacos without change.
...
Enter money for meal: 54
How many people are eating: 2
$54 buys 18 empanadas and 0 tacos without change.
$54 buys 10 empanadas and 6 tacos without change.
$54 buys 2 empanadas and 12 tacos without change.
```

**PARTICIPATION ACTIVITY**

## 6.10.2: Continue statements.

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

Given:

```
for i in range(5):
    if i < 10:
        continue
    print(i)
```

- 1) The loop will print at least some



output.

- True
- False

2) The loop will iterate only once.

- True
- False



©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

**CHALLENGE  
ACTIVITY**

6.10.1: Enter the output of break and continue.



334598.939404.qx3zqy7

Start

Type the program's output

Input

8

Output

0  
1  
2  
12

1

2

3

4

Check

Next

**CHALLENGE  
ACTIVITY**

6.10.2: Simon says.

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4



"Simon Says" is a memory game where "Simon" outputs a sequence of 10 characters (R, G, B, Y) and the user must repeat the sequence. Create a for loop that compares each character of the two strings. For each matching character, add one point to user\_score. Upon a mismatch, end the loop.

Sample output with inputs: 'RRGBRYYBGY' 'RRGBBRYBGY'

User score: 4

334598.939404.qx3zqy7

©zyBooks 09/27/22 11:37 469702

Steven Cameron  
WGUC859v4

```
1 user_score = 0
2 simon_pattern = input()
3 user_pattern  = input()
4
5 ''' Your solution goes here '''
6
7 print('User score:', user_score)
```

Run

View your last submission ▾

## 6.11 Loop else

### Loop else construct

A loop may optionally include an else clause that executes only if the loop terminates normally, not using a break statement. Thus, the complete forms of while and for loops are:

©zyBooks 09/27/22 11:37 469702

Steven Cameron  
WGUC859v4

## Construct 6.11.1: While loop else.

```
while expression: # Loop expression
    # Loop body: Sub-statements to execute if
    # the expression evaluated to True
else:
    # Else body: Sub-statements to execute
once
    # if the expression evaluated to False
# Statements to execute after the loop
```

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

## Construct 6.11.2: For loop else.

```
for name in iterable:
    # Loop body: Sub-statements to
execute
    # for each item in iterable
else:
    # Else body: Sub-statements to
execute
    # once when loop completes
# Statements to execute after the loop
```

The **loop else** construct executes if the loop completes normally. In the following example, a special message "All names printed" is displayed if the entire list of names is completely iterated through.

Figure 6.11.1: Loop else branch taken if loop completes normally.

```
names = ['Janice', 'Clarice', 'Martin', 'Veronica',
'Jason']

num = int(input('Enter number of names to print: '))
for i in range(len(names)):
    if i == num:
        break
    print(names[i], end= ' ')
else:
    print('All names printed.')
```

Enter number of names to print:  
2  
Janice Clarice  
...  
Enter number of names to print:  
8  
Janice Clarice Martin Veronica  
Jason  
All names printed.

## zyDE 6.11.1: Loop else example: Finding a legal baby name.

The country of Denmark allows parents to pick from around 7,000 names for newborns. Names not on the list must receive special approval from the Names Investigation Department of Copenhagen University. (Surprisingly, many countries have naming laws probably to avoid names like "Brfxxccxmnpccccllmmnprxvclmnckssqlbb11116", probably "Albin".)

©zyBooks 09/27/22 11:37 469702

Steven Cameron

WGUC859v4

The program below checks if a user-entered name is an appropriate Danish name. If it is not found in the list of legal names, then a suggestion is made to a close match. If no matches are found, the loop else clause informs the user. Note that the program uses a function called `edit_distance`, which calculates *string edit distance*, or how many characters are different between two strings. For example, the edit distance of "DOG" and "DIG" is 1 because changing the middle character would make the strings equivalent.

Run the program below.

1. Enter the acceptable name "Bjork".
2. Try the name "Michaal", which is not an acceptable name – the program will suggest a replacement based on the edit distance.
3. Try the name "Zoidberg", which is not close at all to any acceptable Danish name. The program will print a special message and terminate.

Current file: [main.py](#) ▾

[Load default template](#)

```
1 import edit_distance
2
3 #A few Legal, acceptable Danish names
4 legal_names = ['Thor', 'Bjork', 'Bailey', 'Anders', 'Bent', 'Bjarne', 'Bente',
5                 'Claus', 'Emil', 'Finn', 'Jakob', 'Karen', 'Julie', 'Johanne', 'Ann',
6                 'Eva', 'Helene', 'Ida', 'Inge', 'Susanne', 'Sofie', 'Rikki',
7                 'Torben', 'Soren', 'Rune', 'Rasmus', 'Per', 'Michael', 'Mads', 'Hans',
8                 'Dorte'
9 ]
10
11 user_name = input('Enter desired name:\n')
12 if user_name in legal_names:
13     print('{} is an acceptable Danish name. Congratulations!'.format(user_name))
14 else:
15     print('{} is not acceptable.'.format(user_name))
16     for name in legal_names:
17         if edit_distance.distance(name, user_name) < 2:
```

Bjork

**Run**

**PARTICIPATION  
ACTIVITY**

## 6.11.1: Loop else.



```
x = 0
y = 5
z = ?
while x < y:
    if x == z:
        print('x == z')
        break
    x += 1
else:
    print('x == y')
```

©zyBooks 09/27/22 11:37 469702

Steven Cameron  
WGUC859v4

- 1) What is the output of the code if z is 3?



- x == z
- x == y

- 2) What is the output of the code if z is 10?



- x == z
- x == y

**CHALLENGE  
ACTIVITY**

## 6.11.1: Loop else.



334598.939404.qx3zqy7

**Start**

Type the program's output

```
result = 0
n = 2
while n < 5:
    print(n, end=' ')
    result += 2
    n += 1
else:
    print('\n {}'.format(result))
print('done')
```

©zyBooks 09/27/22 11:37 469702

Steven Cameron  
WGUC859v4  
2 3 4 \ 6  
done

1

2

3

[Check](#)[Next](#)

## 6.12 Getting both index and value when looping: enumerate()

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

### The enumerate() function

A programmer commonly requires both the current position index and corresponding element value when iterating over a sequence. The example below demonstrates how using a for loop with range() and len() to iterate over a sequence generates a position index but requires extra code to retrieve a value.

Figure 6.12.1: Using range() and len() to iterate over a sequence.

```
origins = [4, 8, 10]
for index in range(len(origins)):
    value = origins[index] # Retrieve value of element in
list.
    print('Element {}: {}'.format(index, value))
```

Element 0: 4  
Element 1: 8  
Element 2:  
10

Similarly, a for loop that iterates over a container obtains the value directly, but must look up the index with a function call.

Figure 6.12.2: Using list.index() to find the index of each element.

```
origins = [4, 8, 10]
for value in origins:
    index = origins.index(value) # Retrieve index of value in
list.
    print('Element {}: {}'.format(index, value))
```

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

Element 0: 4  
Element 1: 8  
Element 2:  
10

The **enumerate()** function retrieves both the index and corresponding element value at the same time, providing a cleaner and more readable solution.

Figure 6.12.3: The enumerate() function.

```
origins = [4, 8, 10]
for (index, value) in enumerate(origins):
    print('Element {}: {}'.format(index, value))
```

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

Element 0: 4  
Element 1: 8  
Element 2:  
10

The `enumerate()` function yields a new tuple each iteration of the loop, with the tuple containing the current index and corresponding element value. In the example above, the `for` loop *unpacks* the tuple yielded by each iteration of `enumerate(origins)` into two new variables: "index" and "value". **Unpacking** is a process that performs multiple assignments at once, binding comma-separated names on the left to the elements of a sequence on the right. Ex: `num1, num2 = [350, 400]` is equivalent to the statements `num1 = 350` and `num2 = 400`.

PARTICIPATION  
ACTIVITY

6.12.1: `enumerate()`.



Use the following code to answer the question below:

```
for (index, value) in enumerate(my_list):
    print(index, value)
```

- 1) If `my_list = ['Greek', 'Nordic', 'Mayan']`, what is the output of the program?



- Greek
- Nordic
- Mayan
- 0 Greek
- 1 Nordic
- 2 Mayan
- 1 Greek
- 2 Nordic
- 3 Mayan

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

**CHALLENGE  
ACTIVITY**

## 6.12.1: Using enumerate in for loops.



334598.939404.qx3zqy7

**Start**

Type the program's output

22 11:37 469702  
Steven Cameron  
WGUC859v4

```
colors = ['yellow', 'red', 'black', 'green', 'blue']
for (position, color) in enumerate(colors):
    print(color, position)
```

```
yellow 0
red 1
black 2
green 3
blue 4
```

**1**

2

**Check****Next**

## 6.13 Additional practice: Dice statistics

The following is a sample programming lab activity; not all classes using a zyBook require students to fully complete this activity. No auto-checking is performed. Users planning to fully complete this program may consider first developing their code in a separate programming environment.

Analyzing dice rolls is a common example in understanding probability and statistics. The following program calculates the number of times the sum of two dice (randomly rolled) is equal to six or seven.

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

## zyDE 6.13.1: Dice statistics.

Load default template...

```
1 import random
2
3 num_sixes = 0
4 num_sevens = 0
5 num_rolls = int(input('Enter number
6
7 if num_rolls >= 1:
8     for i in range(num_rolls):
9         die1 = random.randint(1,6)
10        die2 = random.randint(1,6)
11        roll_total = die1 + die2
12
13        #Count number of sixes and s
14        if roll_total == 6:
15            num_sixes = num_sixes +
16        if roll_total == 7:
17            num_sevens = num_sevens
```

6

Run

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

Create a different version of the program that:

1. Calculates the number of times the sum of the randomly rolled dice equals each possible value from 2 to 12.
2. Repeatedly asks the user for the number of times to roll the dice, quitting only when the user-entered number is less than 1. Hint: Use a while loop that will execute as long as num\_rolls is greater than 1.
3. Prints a histogram in which the total number of times the dice rolls equals each possible value is displayed by printing a character, such as \*, that number of times. The following provides an example:

Dice roll histogram:

```
2s: **
3s: ****
4s: ***
5s: *****
6s: *****
7s: *****
8s: *****
9s: *****
10s: *****
11s: ****
12s: **
```

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

## 6.14 LAB: Convert to binary

Write a program that takes in a positive integer as input, and outputs a string of 1's and 0's representing the integer in binary. For an integer  $x$ , the algorithm is:

```
As long as x is greater than 0
    Output x modulo 2 (remainder is either 0 or 1)
    Assign x with x divided by 2
```

Note: The above algorithm outputs the 0's and 1's in reverse order.

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

Ex: If the input is:

```
6
```

the output is:

```
011
```

6 in binary is 110; the algorithm outputs the bits in reverse.

334598.939404.qx3zqy7

**LAB  
ACTIVITY**

6.14.1: LAB: Convert to binary

10 / 10

main.py

[Load default template...](#)

```
1 user_num = int(input())
2 while user_num >= 1:
3     print(user_num % 2, end='')
4     user_num = user_num // 2
5 print()
```

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

**Develop mode**

**Submit mode**

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

## Enter program input (optional)

If your code requires input values, provide them here.

**Run program**

Input (from above)

**main.py**  
(Your program)

0

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

Program output displayed here

Coding trail of your work [What is this?](#)

8/27 S--0,10 min:5

## 6.15 LAB: Password modifier

Many user-created passwords are simple and easy to guess. Write a program that takes a simple password and makes it stronger by replacing characters using the key below, and by appending "!" to the end of the input string.

- i becomes 1
- a becomes @
- m becomes M
- B becomes 8
- s becomes \$

Ex: If the input is:

mypassword

the output is:

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

Myp@\$\$word!

*Hint: Python strings are immutable, but support string concatenation. Store and build the stronger password in the given **password** variable.*

334598.939404.qx3zqy7

LAB  
ACTIVITY

## 6.15.1: LAB: Password modifier

10 / 10

main.py

[Load default template...](#)

```
1 word = input()
2 password = ''
3
4 for letter in range(len(word)):
5     if word[letter] == 'i':
6         print('1', end='')
7     elif word[letter] == 'a':
8         print('@', end='')
9     elif word[letter] == 'm':
10        print('M', end='')
11    elif word[letter] == 'B':
12        print('8', end='')
13    elif word[letter] == 's':
14        print('$', end='')
15    else:
16        print(word[letter], end=' ')
17 print('!')
```

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

[Develop mode](#)[Submit mode](#)

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

[Run program](#)

Input (from above)

**main.py**  
(Your program)

0

Program output displayed here

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

Coding trail of your work [What is this?](#)

8/27 S-0,10 min:3

## 6.16 LAB: Brute force equation solver

Numerous engineering and scientific applications require finding solutions to a set of equations. Ex:  $8x + 7y = 38$  and  $3x - 5y = -1$  have a solution  $x = 3, y = 2$ . Given integer coefficients of two linear equations with variables  $x$  and  $y$ , use brute force to find an integer solution for  $x$  and  $y$  in the range -10 to 10.

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

Ex: If the input is:

```
8
7
38
3
-5
-1
```

Then the output is:

```
x = 3 , y = 2
```

Use this brute force approach:

```
For every value of x from -10 to 10
    For every value of y from -10 to 10
        Check if the current x and y satisfy both equations. If so,
        output the solution, and finish.
```

Ex: If no solution is found, output:

```
There is no solution
```

You can assume the two equations have no more than one solution.

Note: Elegant mathematical techniques exist to solve such linear equations. However, for other kinds of equations or situations, brute force can be handy.

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

334598.939404.qx3zqy7

**LAB  
ACTIVITY**

6.16.1: LAB: Brute force equation solver

10 / 10

main.py

[Load default template...](#)

```
1 ''' Read in first equation  ax + by = c '''
2 ''' Read in second equation  dx + ey = f '''
3 a = int(input())
4 b = int(input())
5 c = int(input())
6 d = int(input())
7 e = int(input())
8 f = int(input())
9 x = -10
10 y = -10
```

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

**Develop mode**

**Submit mode**

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

**Run program**

Input (from above)



**main.py**  
(Your program)



0

Program output displayed here

Coding trail of your work [What is this?](#)

8/27 S-0, 6, 10 min:5

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

## 6.17 LAB: Adjust values in a list by normalizing

When analyzing data sets, such as data for human heights or for human weights, a common step is to adjust the data. This adjustment can be done by normalizing to values between 0 and 1, or throwing away outliers.

For this program, adjust the values by dividing all values by the largest value. The input begins with an integer indicating the number of floating-point values that follow.

Output each floating-point value with two digits after the decimal point, which can be achieved as follows:

```
print('{:.2f}'.format(your_value))
```

Ex: If the input is:

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

```
5
30.0
50.0
10.0
100.0
65.0
```

the output is:

```
0.30
0.50
0.10
1.00
0.65
```

The 5 indicates that there are five floating-point values in the list, namely 30.0, 50.0, 10.0, 100.0, and 65.0. 100.0 is the largest value in the list, so each value is divided by 100.0.

334598.939404.qx3zqy7

**LAB ACTIVITY**

6.17.1: LAB: Adjust values in a list by normalizing

10 / 10

main.py

[Load default template...](#)

```
1 max = 0
2 listA = []
3 # Input number of elements
4 n = int(input())
5
6 for i in range(0, n):
7     #print("Enter element No-{}: ".format(i + 1), end=' ')
8     listA.append(float(input())) # adding the element
9     if listA[i] > max:
10         max = listA[i]
11
12 for i in listA:
13     print('{:.2f}'.format(i / max))
```

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4

**Develop mode****Submit mode**

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

©zyBooks 09/27/22 11:37 469702Steven Cameron  
WGUC859v4**Enter program input (optional)**

If your code requires input values, provide them here.

**Run program**

Input (from above)

**main.py**  
(Your program)

0

**Program output displayed here**Coding trail of your work    [What is this?](#)

8/28 U-0,10 min:8

©zyBooks 09/27/22 11:37 469702  
Steven Cameron  
WGUC859v4