5.1 Loops (general)

Loop concept

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

PARTICIPATION ACTIVITY 5.1.1: 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 5.1.2: 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

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

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 5.1.3: 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.

PARTICIPATION ACTIVITY 5.1.4: 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 5.1.5: Loop example: Average.

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

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

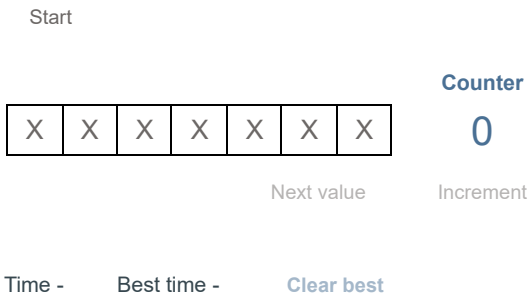
Example: Counting specific values in a list

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 5.1.6: Counting negative values in a list of values.

Click "Increment" if a negative value is seen.



PARTICIPATION ACTIVITY 5.1.7: Counting negative values.

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?

- ○ val > 0
- ○ val < 0
- ○ val is 0

2) What should statement (B) be?

- ○ val = val + 1

count = count + 1

- ○ count = val

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

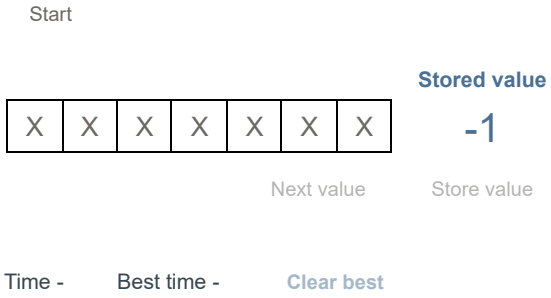
- ○ Yes
- ○ No

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 5.1.8: Find the maximum value in the list of values.

Click "Store value" if a new maximum value is seen.



PARTICIPATION ACTIVITY 5.1.9: Determining the max value.

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?

- ○ -1, 20
- ○ -1, 5, 10, 20
- ○ -1, 5, 10, 7, 20

5.2 While loops

While loop: Basics

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 5.2.1: 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 5.2.1: 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.
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 5.2.2: Basic while loops.

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

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]`.

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

Figure 5.2.1: 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 5.2.3: Loop expressions.

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

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
```

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.

PARTICIPATION ACTIVITY 5.2.4: 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.

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.

zyDE 5.2.1: While loop example: Ancestors printing program.

Run the program below.

Load default templat

```
year_considered = 2020  # Year being considered
num_ancestors = 2  # Approx. ancestors in considered year
years_per_generation = 20  # Approx. years per generation

user_year = int(input('Enter a past year (neg. for B.C.): '))
print()

while year_considered >= user_year:
    print('Ancestors in {}: {}'.format(year_considered, num_ancestors))

    num_ancestors = num_ancestors * 2
    year_considered = year_considered - years_per_generation

```


1945

Run

PARTICIPATION ACTIVITY 5.2.5: 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')
```

Check Show answer

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

Check Show answer

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 5.2.1: Enter the output of the while loop.

Start

Type the program's output.

```
g = 0

while g <= 1:
    print(g, end=' ')
    g += 1
```

1 2 3 4

Check Next

View solution *(Instructors only)*

CHALLENGE ACTIVITY 5.2.2: Basic while loop with user input.

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.
```

```
user_num = int(input())
while ''' Your solution goes here ''':
  print('Body')
  user_num = int(input())

print('Done.')
```

Run

View solution *(Instructors only)*

Download student submissions

CHALLENGE ACTIVITY 5.2.3: Basic while loop expression.

Write a while loop that prints user_num divided by 2 until user_num is less than 1. The value of user_num changes inside of the loop.

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.

```
user_num = int(input())

''' Your solution goes here '''
```


Run

View solution *(Instructors only)*

Download student submissions

5.3 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 5.3.1: While loop example: GCD program.

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

Load default template...

```
num_a = int(input('Enter first positive integ
print()

num_b = int(input('Enter second positive inte
print()

while num_a != num_b:
    if num_a > num_b:
        num_a = num_a - num_b
    else:
        num_b = num_b - num_a

print('GCD is {}'.format(num_a))
```

```
20
15
```

Run

PARTICIPATION ACTIVITY 5.3.1: Loop example: Greatest common divisor.

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?

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 ***multi-line comment***, which is delimited at the beginning and end by triple-quotes. Either single ' or double " quotes can be used.

Construct 5.3.1: Multi-line comments.

```
"""
This is a multi-line comment
that continues on a second line.
And even uses a third line.
"""
```

zyDE 5.3.2: While loop example: Conversation.

Run the program below. Try adding additional conditions to leave the conversation, such "See you later".

Load default templa

```
Thank you very much, Mr. Robot.
Goodbye
```

Run

```
'''
Program that has a conversation with the user.
Uses elif branching and a random number to mix up the program's responses.
'''
import random  # Import a library to generate random numbers

print('Tell me something about yourself.')
print('You can type \'Goodbye\' at anytime to quit.\n')
user_text = input()

while user_text != 'Goodbye':
    random_num = random.randint(0, 2)  # Gives a random number between 0 and 2
    if random_num == 0:
        print('\nPlease explain further.\n')
    elif random_num == 1:
        print("\nWhy do you say: '{}'?\n".format(user_text))
    elif random_num == 2:
        print('\nWhat else can you share?\n')
    else:
        print('\nMultiple statements, try again.\n')
```

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 contents will execute. Within the while loop, 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 5.3.2: 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

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

Check Show answer

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

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.

zyDE 5.3.3: Computing average of a list with a sentinel.

Load default templat

```
'''
Outputs average of list of positive integers
List ends with 0 (sentinel)
Ex: 10 1 6 3 0  yields (10 + 1 + 6 + 3) / 4, or 5
'''

values_sum = 0
num_values = 0

curr_value = int(input())

while curr_value > 0: # Get values until 0 (or less)
    values_sum += curr_value
    num_values += 1
    curr_value = int(input())

print('Average: {:.0f}\n'.format(values_sum / num_values))
```



```
10
1
6
```

Run

PARTICIPATION ACTIVITY 5.3.3: 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

2) For the given input sequence, what is the final value of num_values?
   - ○ 0
   - ○ 4
   - ○ 5

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

CHALLENGE ACTIVITY 5.3.1: While loop with sentinel.

Start

Type the program's output.


```
entered_value = int(input())

maximum_number = entered_value

while entered_value > 0:
    if entered_value > maximum_number:
        maximum_number = entered_value
    entered_value = int(input())

print('Max value:', maximum_number, end='')
```

Input

```
44
23
56
12
0
```

Output

1 2

Check Next

View solution *(Instructors only)*

CHALLENGE ACTIVITY 5.3.2: 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?
```

```
1  import random
2  random.seed(5)
3
4  keep_going = '-'
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_going = input()
```


Run

View solution *(Instructors only)*

Download student submissions

CHALLENGE ACTIVITY 5.3.3: While loop: Insect growth.

Given positive integer num_insects, write a while loop that prints that number doubled up to, but without exceeding 100. Follow each number with a space.

Sample output with input: 8

```
8 16 32 64
```

```
num_insects = int(input()) # Must be >= 1

''' Your solution goes here '''

```

Run

View solution *(Instructors only)*

Download student submissions

5.4 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[1] with the following form is used:

Construct 5.4.1: Counting while loop form.

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

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 5.4.1: While loop that counts iterations: Savings interest program.

Load default templat


```
'''Program that calculates savings and interest'''

initial_savings = 10000
interest_rate = 0.05

print('Initial savings of ${}'.format(initial_savings))
print('at {:.0f}% yearly interest.\n'.format(interest_rate*100))

years = int(input('Enter years: '))
print()

savings = initial_savings
i = 1  # Loop variable
while i <= years:  # Loop condition
    print(' Savings at beginning of year {}: ${:.2f}'.format(i, savings))
    savings = savings + (savings*interest_rate)
    i = i + 1  # Increment loop variable

print('\n')
```

10

Run

PARTICIPATION ACTIVITY 5.4.1: Savings interest program.

Refer to the program above.

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

Check Show answer

2) If interest_rate is 3% and initial_savings are $5000, savings > $7500 after how many loop iterations?

Check Show answer

PARTICIPATION ACTIVITY 5.4.2: Basic counting with while loops.

Use <= in each answer.

1) Loop iterates 10 times.

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

Check Show answer

2) Loop iterates 99 times.

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

Check Show answer

3) Loop iterates 2 times.

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

Check Show answer

Other forms of counting

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 5.4.1: 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 5.4.2: Loop variable increased by 2 per iteration.

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

zyDE 5.4.2: Loop over presidential election years.

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

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

Load default template... **Run**


```
year = 1792
current_year = ?

while year ? ??:
    # Print the election year
    year = year + ?

```

PARTICIPATION ACTIVITY 5.4.3: 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
```

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).

```
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 5.4.4: 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.

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

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

zyDE 5.4.3: Calculate a factorial.

Write a program that lets a user enter N and that outputs N! (N factorial, meaning N*(N-1 2)*...*2*1). Hint: Initialize a variable total to N (where N is input), and use a loop variable i counts from total-1 down to 1. Compare your output with some of these answers: 1:1, 2:2 4:24, 5:120, 8:40320.

Load default template...

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

5

Run

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 5.4.2: Operators like += are common in loops.

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


PARTICIPATION ACTIVITY 5.4.5: 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 5.4.1: 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
```

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

Run

View solution *(Instructors only)*

Download student submissions

CHALLENGE ACTIVITY 5.4.2: Printing output using a counter.

Retype and run, note incorrect behavior. Then fix errors in the code, which should print num_stars asterisks.

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

Sample output with input: 3

```
*
*
*
```

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

Run

View solution *(Instructors only)*

Download student submissions

(*1) To instructors: Focus is placed on mastering basic looping using while loops, before introducing for loops and range(). Also, looping N times is initially done using 1 <= N rather than 0 < N.

5.5 For loops

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 the next element to a variable 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 next element in the container to the name given in the for loop statement.

Construct 5.5.1

```
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 5.5.1: Iterating over a list using a for loop.

Animation content:

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 to 'Bill', the second iteration assigns name to 'Nicole', and the final iteration assigns name to '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 *keys* of the dictionary to the loop variable. The values can then be accessed using the key.

Figure 5.5.1: A for loop assigns a dictionary's keys to the loop variable.

```
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 next character of the string to the loop variable. Strings are sequence types just like lists, so the behavior is identical (leftmost character first, then each following character).

Figure 5.5.2: 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_._
```

PARTICIPATION ACTIVITY 5.5.2: 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

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.

Figure 5.5.3: 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))
```

```
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 5.5.4: 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 |
```


PARTICIPATION ACTIVITY 5.5.3: 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)
```

Check Show answer

2) Assign num_neg to 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.

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

for scr in :
    print(scr, end=' ')
```

Check Show answer

CHALLENGE ACTIVITY 5.5.1: 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
```



```
# NOTE: The following statement converts the input into a list container
stock_prices = input().split()

for ''' Your solution goes here ''':
    print('$', price)
```

Run

View solution *(Instructors only)*

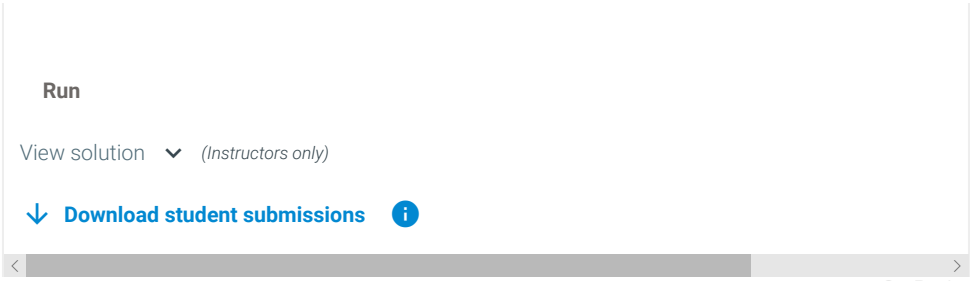
Download student submissions

CHALLENGE ACTIVITY 5.5.2: 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
```



5.6 Counting using the range() function

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 numbers, starting at zero and ending before a value given inside the parentheses. Ex: `for i in range(3)` sets i to 0 during the first iteration of the for loop, i to 1 during the second iteration, and finally i to 2 on the third iteration. The value within the parentheses is not included in the generated sequence.

The range() function can take up to three arguments to indicate the starting value of the sequence, the ending value of the sequence minus 1, and the interval between numbers.

Table 5.6.1: Using the range() function.

Range	Generated sequence	Explanation
`range(5)`	`0 1 2 3 4`	Every integer from 0 to 4.
`range(0, 5)`	`0 1 2 3 4`	Every integer from 0 to 4.
`range(3, 7)`	`3 4 5 6`	Every integer from 3 to 6.
`range(10, 13)`	`10 11 12`	Every integer from 10 to 12.
`range(0, 5, 1)`	`0 1 2 3 4`	Every 1 integer from 0 to 4.
`range(0, 5, 2)`	`0 2 4`	Every 2nd integer from 0 to 4.
`range(5, 0, -1)`	`5 4 3 2 1`	Every 1 integer from 5 down to 1
`range(5, 0, -2)`	`5 3 1`	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 5.6.1: For loop example: Calculating yearly savings.

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

Load default template...

```
'''Program that calculates savings and intere

initial_savings = 10000
interest_rate = 0.05

years = int(input('Enter years: '))
print()

savings = initial_savings
for i in range(years):
    print(' Savings in year {}: ${:.2f}'.form
    savings = savings + (savings*interest_rat

print('\n')
```

8

Run

PARTICIPATION ACTIVITY 5.6.1: 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

PARTICIPATION ACTIVITY 5.6.2: 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

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 5.6.1: Enter the for loop's output.

5.7 While vs. for loops

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 5.7.1: 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 5.7.2: While loops and for loops.

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.
   - ○ while
   - ○ for

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

5.8 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'.

Figure 5.8.1: 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:
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.)

zyDE 5.8.1: Two character dotcom domain names.

Modify the program to include two-character .com names, where the second character c 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...

Run

zyDE 5.8.2: 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 units. So if the user enters 40, print 8 asterisks.

```
'''
Program to print all 2-letter domain names.
Note that ord() and chr() convert between tex
ord('a') is 97. ord('b') is 98, and so on. ch
'''
print('Two-letter domain names:')

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


num = 0
while num >= 0:
    num = int(input('Enter an integer (negati


    if num >= 0:
        print('Depicted graphically:')
        for i in range(num):
            print('*', end='')
        print('\n')

print('Goodbye.')
```

Load default template...

```
5
10
-1
```

Run

PARTICIPATION ACTIVITY 5.8.1: 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

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

```
for i in range(5):
    for j in range(10, 12):
        print('{}{}'.format(i, j))
```

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

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 5.8.1: Nested loops: Print rectangle

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

Sample output with inputs: 2 3

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

```
num_rows = int(input())
num_cols = int(input())

''' Your solution goes here '''

        print('*', end=' ')
    print()
```


Run

View solution *(Instructors only)*

Download student submissions

CHALLENGE ACTIVITY 5.8.2: 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
```



```
num_rows = int(input())
num_cols = int(input())

# Note 1: You will need to declare more variables
# Note 2: Place end=' ' at the end of your print statement to separate seats by spaces

''' Your solution goes here '''

print()
```

Run

View solution *(Instructors only)*

Download student submissions

5.9 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.

Figure 5.9.1: 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 5.9.2: 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????????
```

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.

Figure 5.9.3: 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))
```

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


The fourth version can be created by filling in the if-else 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 5.9.4: 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????
```


zyDE 5.9.1: Complete the phone number program.

Complete the program by providing the additional if-else branches for decoding other lett
a phone number. Try incrementally writing the program by adding one "else if" branch at a
time, testing that each added branch works as intended.

Load default template...

```
1  user_input = input('Enter phone number:\n')
2  phone_number = ''
3
4  for character in user_input:
5      if ('0' <= character <= '9') or (characte
6          phone_number += character
7      elif ('a' <= character <= 'c') or ('A' <=
8          phone_number += '2'
9      #FIXME: Add remaining elif branches
10     else:
11         phone_number += '?'
12
13 print('Numbers only: {}'.format(phone_number)
14
```

1-555-HOLIDAY

Run

PARTICIPATION ACTIVITY 5.9.1: 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.
   - ○ True
   - ○ False

5.10 Break and continue

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.

Figure 5.10.1: 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.')
```

```
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 5.10.1: Break statements.

Given the following while loop, what is the value assigned to variable z 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
```

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

Check Show answer

2) a = 4, b = 5, c = 20

Check Show answer

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.

Figure 5.10.2: 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.
```

ACTIVITY 5.10.2: Continue statements.

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

CHALLENGE ACTIVITY 5.10.1: Simon says.

"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 the two strings. For each match, add one point to user_score. Upon a mismatch, end the game.

Sample output with inputs: 'RRGBRYYBGY' 'RRGBBRYBGY'

```
User score: 4
```

```
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 solution *(Instructors only)*

Download student submissions

5.11 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:

Construct 5.11.1: 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
```


Construct 5.11.2: 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 5.11.1: 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 5.11.1: Loop else example: Finding a legal baby name.

The country of Denmark allows parents to pick from around 7,000 names for newborn in 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 "Brfxxccxxmnpcccclllmmnprxvclmnckssqlbb11116" (pronou "Albin").]

The program below checks if a user-entered name is an appropriate Danish name. If the is not found in the list of legal names, then a suggestion is made to a close match. If no c matches are found, the loop else clause informs the user. Note that the program uses the third-party module called edit_distance, which calculates *string edit distance*, or how mar 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 sugge replacement based on the edit distance.
3. Try the name "Zoidberg", which is not close at all to any acceptable Danish names - program will print a special message and terminate.

Load default templa

```
import edit_distance

#A few legal, acceptable Danish names
legal_names = ['Thor', 'Bjork', 'Bailey', 'Anders', 'Bent', 'Bjarne', 'Bjorn',
    'Claus', 'Emil', 'Finn', 'Jakob', 'Karen', 'Julie', 'Johanne', 'Anna', 'Anne',
    'Bente', 'Eva', 'Helene', 'Ida', 'Inge', 'Susanne', 'Sofie', 'Rikkie', 'Pia',
    'Torben', 'Soren', 'Rune', 'Rasmus', 'Per', 'Michael', 'Mads', 'Hanne',
    'Dorte'
]

user_name = input('Enter desired name:\n')
if user_name in legal_names:
    print('{} is an acceptable Danish name. Congratulations.'.format(user_name))
else:
    print('{} is not acceptable.'.format(user_name))
    for name in legal_names:
        if edit_distance.distance(name, user_name)  < 2:
            print('You might consider: {},'.format(name), end=' ')
            break
    else:
```


Bjork

Run

PARTICIPATION ACTIVITY 5.11.1: Loop else.

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

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

5.12 Getting both index and value when looping: enumerate()

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 5.12.1: 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 5.12.2: 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))
```

```
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 5.12.3: The enumerate() function.

```
origins = [4, 8, 10]

for (index, value) in enumerate(origins):
    print('Element {}: {}'.format(index, value))
```

```
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 5.12.1: enumerate().

Use the following code to answer the questions 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

5.13 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.

zyDE 5.13.1: Dice statistics.

Load default template...

```
import random

num_sixes = 0
num_sevens = 0
num_rolls = int(input('Enter number of roll

if num_rolls >= 1:
    for i in range(num_rolls):
        die1 = random.randint(1,6)
        die2 = random.randint(1,6)
        roll_total = die1 + die2

        #Count number of sixes and sevens
        if roll_total == 6:
            num_sixes = num_sixes + 1
        if roll_total == 7:
            num_sevens = num_sevens + 1
        print('Roll {} is {} ({} + {})'.fo
```

6

Run

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: **
```


5.14 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.

Ex: If the input is:

```
6
```

the output is:

```
011
```

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

LAB ACTIVITY 5.14.1: LAB: Convert to binary 0 / 10

main.py Load default template...

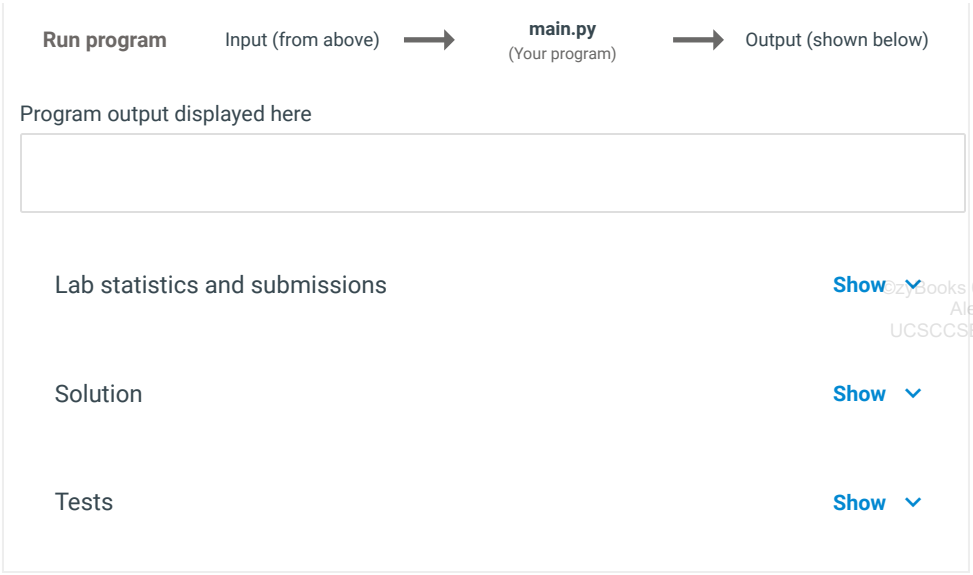
```
1 ''' Type your code here. '''
2
```


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.



5.15 LAB: Count input length without spaces, periods, or commas

Given a line of text as input, output the number of characters excluding spaces, periods, or commas.

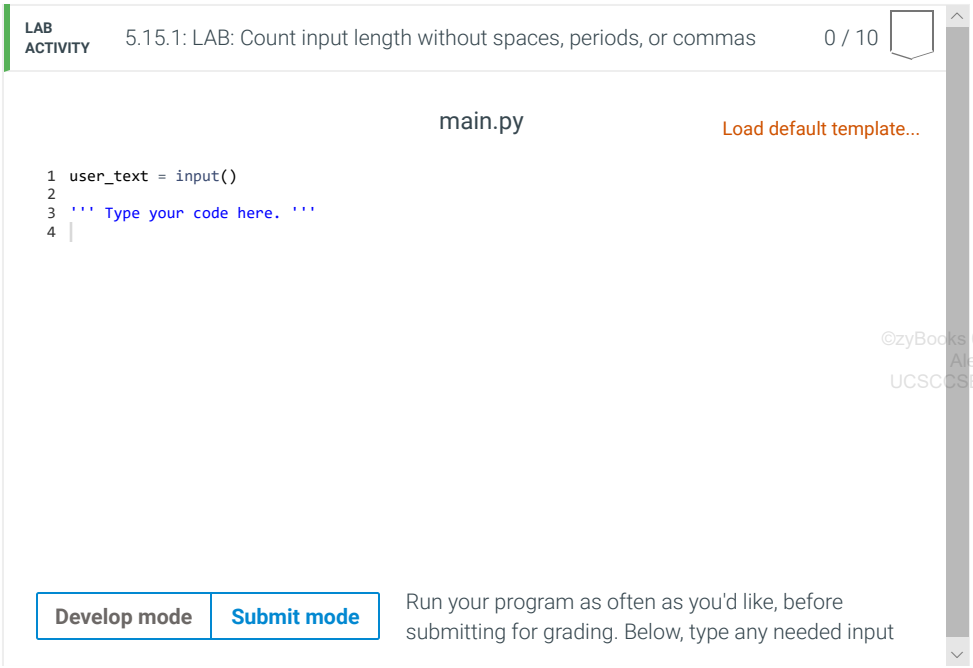
Ex: If the input is:

```
Listen, Mr. Jones, calm down.
```

the output is:

```
21
```

Note: Account for all characters that aren't spaces, periods, or commas (Ex: "r", "2", "!").



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) → Output (shown below)

Program output displayed here

Lab statistics and submissions **Show**

Solution **Show**

Tests **Show**

5.16 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 "q*s" to the end of the input string.

- i becomes !
- a becomes @
- m becomes M
- B becomes 8
- o becomes .

Ex: If the input is:

```
mypassword
```

the output is:

```
Myp@ssw.rdq*s
```

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

LAB ACTIVITY 5.16.1: LAB: Password modifier 0 / 10

main.py Load default template...

```
word = input()
password = ''

''' Type your code here. '''
```


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) → Output (shown below)

Program output displayed here

Lab statistics and submissions — Show

Solution — Show

Tests — Show

5.17 LAB: Output range with increment of 10

Write a program whose input is two integers. Output the first integer and subsequent increments of 10 as long as the value is less than or equal to the second integer.

Ex: If the input is:

```
-15
30
```


the output is:

```
-15 -5 5 15 25
```

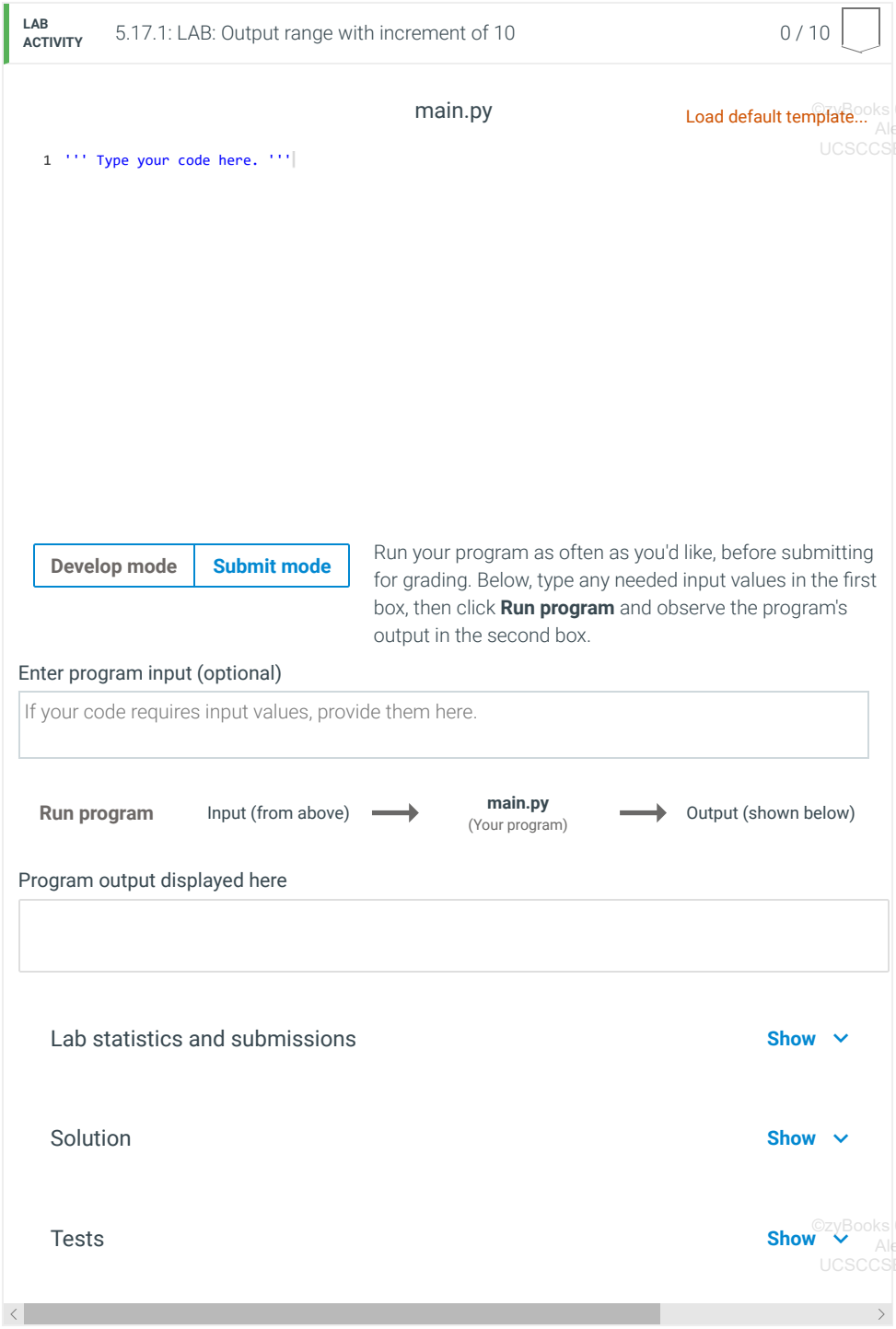
Ex: If the second integer is less than the first as in:

```
20
5
```

the output is:

```
Second integer can't be less than the first.
```

For coding simplicity, output a space after every integer, including the last.



5.18 LAB: Print string in reverse

Write a program that takes in a line of text as input, and outputs that line of text in reverse. The program repeats, ending when the user enters "Quit", "quit", or "q" for the line of text.

Ex: If the input is:

```
Hello there
Hey
quit
```

then the output is:


```
ereht olleH
yeH
```

LAB ACTIVITY 5.18.1: LAB: Print string in reverse 0 / 10

main.py Load default template...

```
1 ''' Type your code here. '''
2
```

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) → Output (shown below)

Program output displayed here

Lab statistics and submissions Show

Solution Show

Tests Show

5.19 LAB: Countdown until matching digits

Write a program that takes in an integer in the range 20-98 as input. The output is a countdown starting from the integer, and stopping when both output digits are identical.

Ex: If the input is:

```
93
```

the output is:

```
93
92
91
90
89
88
```

Ex: If the input is:

```
77
```

the output is:

```
77
```

Ex: If the input is:

```
15
```

or any value not between 20 and 98 (inclusive), the output is:

```
Input must be 20-98
```

Use a while loop. Compare the digits; do not write a large if-else for all possible same-digit numbers (11, 22, 33, ..., 88), as that approach would be cumbersome for larger ranges.

LAB ACTIVITY 5.19.1: LAB: Countdown until matching digits 0 / 10

main.py Load default template...

```
1 ''' Type your code here. '''
2
```


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) → Output (shown below)

Program output displayed here

Lab statistics and submissions **Show**

Solution **Show**

Tests **Show**

5.20 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.

Ex: If the input is:

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

Then the output is:

```
3 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:

```
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.

LAB ACTIVITY 5.20.1: LAB: Brute force equation solver 0 / 10

main.py Load default template...


```
''' Read in first equation, ax + by = c '''
a = int(input())
b = int(input())
c = int(input())

''' Read in second equation, dx + ey = f '''
d = int(input())
e = int(input())
f = int(input())

''' Type your code here. '''

```

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) → Output (shown below)

Program output displayed here

Lab statistics and submissions Show

Solution Show

Tests Show

5.21 LAB: Smallest and largest numbers in a list

Write a program that reads a list of integers into a list as long as the integers are greater than zero, then outputs the smallest and largest integers in the list.

Ex: If the input is:

```
10
5
3
21
2
-6
```


the output is:

```
2 21
```

You can assume that the list of integers will have at least 2 values.

LAB ACTIVITY 5.21.1: LAB: Smallest and largest numbers in a list 0 / 10

main.py Load default template...

```
1 ''' Type your code here. '''
2
```

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) → Output (shown below)

Program output displayed here

Lab statistics and submissions Show

Solution Show

Tests Show

5.22 LAB: Output values in a list below a user defined amount

Write a program that first gets a list of integers from input. The input begins with an integer indicating the number of integers that follow. Then, get the last value from the input, which indicates a threshold. Output all integers less than or equal to that last threshold value.

Ex: If the input is:

```
5
50
60
140
200
75
100
```

the output is:

```
50
60
75
```

The 5 indicates that there are five integers in the list, namely 50, 60, 140, 200, and 75. The 100 indicates that the program should output all integers less than or equal to 100, so the program outputs 50, 60, and 75.

Such functionality is common on sites like Amazon, where a user can filter results.

LAB ACTIVITY 5.22.1: LAB: Output values in a list below a user defined amount 0 / 10

main.py Load default template...

```
1 ''' Type your code here. '''
2
```


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) → Output (shown below)

Program output displayed here

Lab statistics and submissions Show

Solution Show

Tests Show

5.23 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 can be done by normalizing to values between 0 and 1, or throwing away outliers.

Write a program that first gets a list of integers from input. The input begins with an integer indicating the number of integers that follow. Then, adjust each integer in the list by subtracting the smallest value from all the integers.

Ex: If the input is:

```
5
30
50
10
70
65
```

the output is:

```
20
40
0
60
55
```


The 5 indicates that there are five integers in the list, namely 30, 50, 10, 70, and 65. The smallest value in the list is 10, so the program subtracts 10 from all integers in the list.

LAB ACTIVITY 5.23.1: LAB: Adjust values in a list by normalizing 0 / 10

main.py Load default template...

```
1 ''' Type your code here. '''
```


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) → Output (shown below)

Program output displayed here

Lab statistics and submissions **Show**

Solution **Show**

Tests **Show**

5.24 LAB: Warm up: Drawing a right triangle

This program will output a right triangle based on user specified height triangle_height and symbol triangle_char.

(1) The given program outputs a fixed-height triangle using a * character. Modify the given program to output a right triangle that instead uses the user-specified triangle_char character. (1 pt)

(2) Modify the program to use a loop to output a right triangle of height triangle_height. The first line will have one user-specified character, such as % or *. Each subsequent line will have one additional user-specified character until the number in the triangle's base reaches triangle_height. Output a space after each user-specified character, including a line's last user-specified character. (2 pts)

Example output for triangle_char = % and triangle_height = 5:

```
Enter a character:
%
Enter triangle height:
5


%
% %
% % %
% % % %
% % % % %
```


LAB ACTIVITY 5.24.1: LAB: Warm up: Drawing a right triangle 0 / 3

main.py Load default template...

```
1 triangle_char = input('Enter a character:\n')
2 triangle_height = int(input('Enter triangle height:\n'))
3 print('')
4
5 print ('* ')
6 print ('* * ')
7 print ('* * * ')
8
```

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) → Output (shown below)

Program output displayed here

Lab statistics and submissions Show

Solution Show

Tests Show

5.25 LAB*: Program: Drawing a half arrow

This program outputs a downwards facing arrow composed of a rectangle and a right triangle. The arrow dimensions are defined by user specified arrow base height, arrow base width, and arrow head width.

(1) Modify the given program to use a loop to output an arrow base of height arrow_base_height. (1 pt)

(2) Modify the given program to use a loop to output an arrow base of width arrow_base_width. (1 pt)

(3) Modify the given program to use a loop to output an arrow head of width arrow_head_width. (2 pts)

(4) Modify the given program to only accept an arrow head width that is larger than the arrow base width. Use a loop to continue prompting the user for an arrow head width until the value is larger than the arrow base width. (1 pt)

```
while arrow_head_width <= arrow_base_width:
    arrow_head_width = int(input('Enter arrow head width:\n'))
```

Example output for arrow_base_height = 5, arrow_base_width = 2, and arrow_head_width = 4:

```
Enter arrow base height:
5
Enter arrow base width:
2
Enter arrow head width:
4

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

LAB ACTIVITY 5.25.1: LAB*: Program: Drawing a half arrow 0 / 5

main.py Load default template...

```
1  arrow_base_height = int(input('Enter arrow base height:\n'))
2
3  arrow_base_width = int(input('Enter arrow base width:\n'))
4
5  arrow_head_width = int(input('Enter arrow head width:\n'))
6
7  print('')
8  # Draw arrow base (height = 3, width = 2)
9  print ('**')
10 print ('**')
11 print ('**')
12
13 # Draw arrow head (width = 4)
14 print ('****')
15 print ('***')
16 print ('**')
17 print ('*')
18
```

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) → Output (shown below)

Program output displayed here

Lab statistics and submissions **Show**

Solution **Show**

Tests **Show**

©zyBooks 03/05/20 10:25 591419
Alexey Munishkin
UCSCCSE20NawabWinter2020