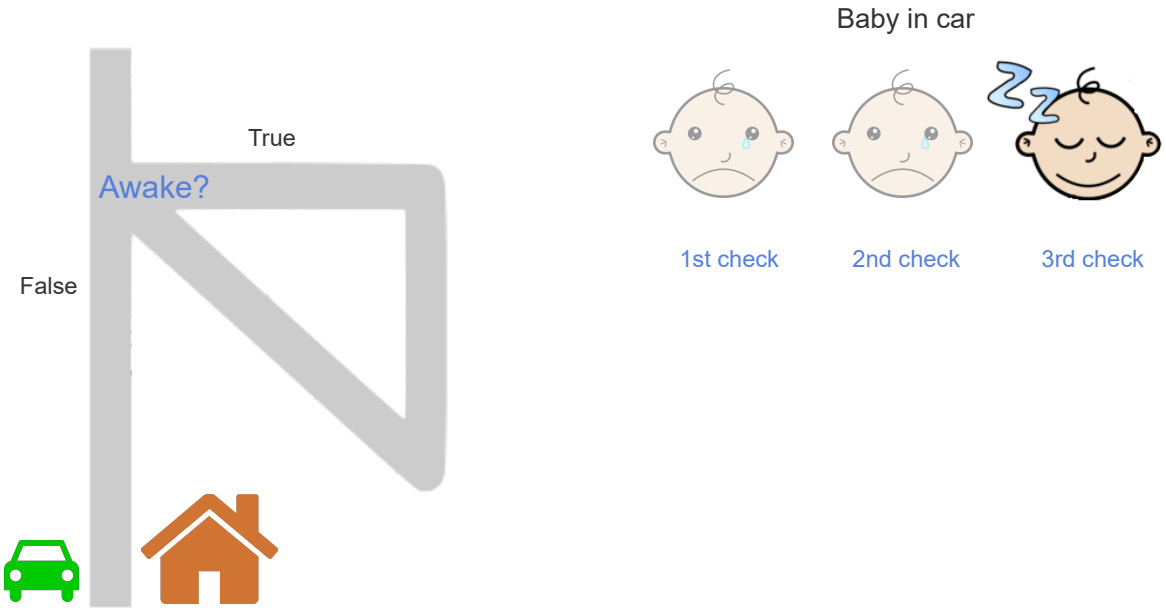
5.1 Loops (general)

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 content:

Step 1: Parents may be familiar with this scenario: Driving home, the baby is awake. Parents circle the block, hoping the baby will fall asleep.
A diagram of a house, car, and a block near the house. There is a line/road between the house and the block. The block is in the shape of a triangle. The car starts at one point of the triangle, which is labelled Awake? The text True appears on one line of the triangle. The text False appears on the line/road going from block/triangle to the house.
The car drives around the triangle, ending at the same point the car started. A picture of a sad baby appears, with the label 1st check below the baby, and the text Baby in car above the sad baby.

Step 2: After the first loop, the baby is still awake, so the parents loop again.
The car drives around the triangle/block again, ending at the same point of the triangle that it started. A second picture of a sad baby appears, with the text 2nd check below the sad baby.

Step 3: After the second loop, the baby is asleep, so the parents head home for a peaceful evening.

The car starts at the same point as before, but this time, drives straight to the house rather than driving around the triangle. The line between the starting point and the house is labelled False. A third picture of a baby appears, but this time the baby is happy and sleepy. The label below the baby is 3rd check.

Animation captions:

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

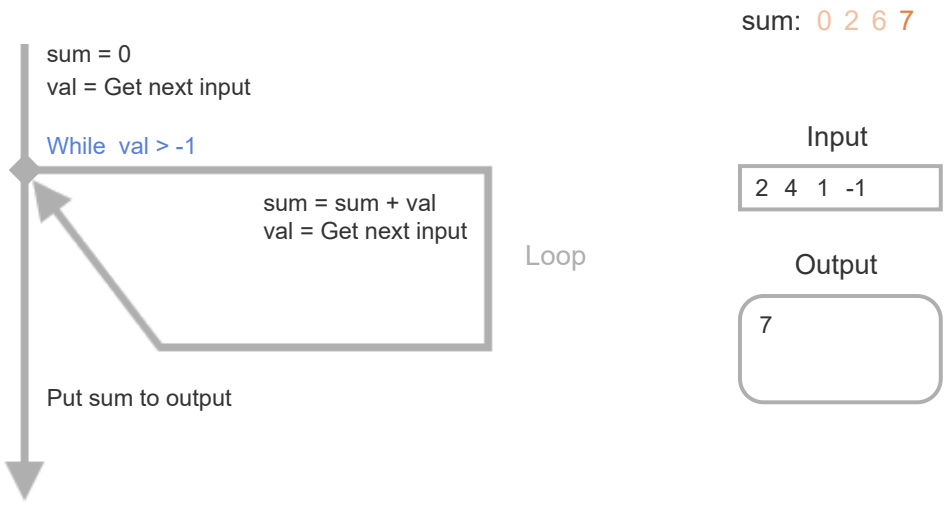
4) How many loops around the block did the parents make?
   - 2
   - 3

5) Where was the decision point for whether to loop: At the top of the street or the 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 the expression is 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 content:

Static Figure:
A diagram with an arrow pointing down. There is a defined diamond near the top of the arrow. A series of lines create a loop starting at the diamond and to the right of the downward pointing arrow. This loop is fairly rectangular, but ends with an arrow pointing back to the diamond.
A program is superimposed on the loop and arrow.
The following code appears at the top of the downward pointing arrow, above the diamond:
sum = 0
val = Get next input
The following code is displayed next to the diamond and the right horizontal line of the loop:
While val > -1
The following code is displayed inside the loop:
sum = sum + val
val = Get next input
The following code is displayed between the diamond and the bottom or point of the downward pointing arrow:
Put sum to output
Next to the diagram, the text Input is displayed, with a box below that contains the numbers: 2 4 1 -1

The output console contains 1 line of output:
7
Next to the diagram, the following text is displayed:
sum: 0 2 6 7
The 0, 2, and 6 are faded, while the 7 is a stronger color.

Step 1: A loop is like a branch, but the loop jumps back to the expression when done. Thus, the loop's statements may execute multiple times before execution proceeds past the loop.
A diagram with an arrow pointing down. There is a defined diamond near the top of the arrow. A series of lines create a loop starting at the diamond and to the right of the downward pointing arrow. This loop is fairly rectangular, but ends with an arrow pointing back to the diamond.
A small square travels down the downward pointing arrow, to the diamond. Then, the square goes around the loop back to the diamond, and then goes around the loop back to the diamond again. Then, the square travels down the downward pointing arrow until the square reaches the bottom/point of the downward pointing arrow.

Step 2: This program receives an input value. If the value > -1, the program adds the value to a sum, receives another input, and repeats. val is 2, so the loop's statements execute, making sum 2.
A program appears superimposed on the loop and arrow described in step 1. Next to the diagram, the text Input appears, with a box below that contains the numbers: 2 4 1 -1
The text Output appears, with an empty box below. This is the output console.
In the diagram, the following code appears at the top of the downward pointing arrow, above the diamond:
sum = 0
val = Get next input
The square starts travelling again. The square travels by sum = 0, and the text sum: 0 appears next to the diagram.
The square travels by val = Get next input, where the value 2 is shown.
The square travels to the diamond, where the text
While val > -1
is displayed. This condition is true, so the square travels along the horizontal line of the loop, going right.
Within the loop, the square travels by the code
sum = sum + val
Then, sum: 2
is displayed beside the loop diagram.
Then, the square travels by the code
val = Get next input
within the loop, where 4 is displayed as the value for val.
Then, the square completes travelling around the loop, and returns to the diamond.

Step 3: The loop's statements ended by receiving 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.
While val > -1

is true, so the square starts travelling the loop again, moving to the right horizontally.
The square travels to the code
sum = sum + val
The text
sum: 6
is displayed next to the loop diagram.
Then, the square travels by the code
val = Get next input
within the loop, where 1 is displayed as the value for val.
Then, the square completes travelling around the loop, and returns to the diamond.

Step 4: The loop's statements receive 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.
While val > -1
is true, so the square starts travelling the loop again, moving to the right horizontally.
The square travels to the code
sum = sum + val
The text
sum: 7
is displayed next to the loop diagram.
Then, the square travels by the code
val = Get next input
within the loop, where -1 is displayed as the value for val.
Then, the square completes travelling around the loop, and returns to the diamond.

Step 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.
The square travels downward from the diamond, down the downward pointing arrow. The square travels by the code
Put sum to output
The output console now contains 1 line of output:
7

Animation captions:

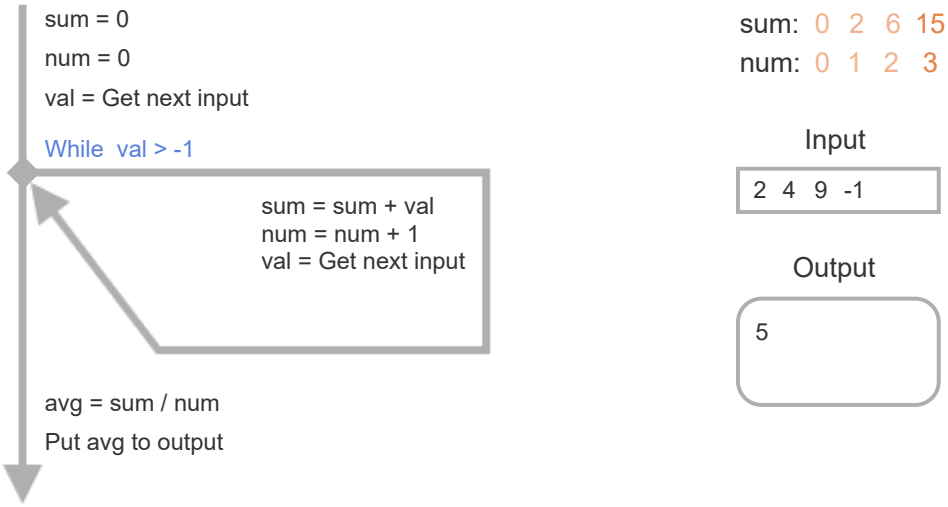
1. A loop is like a branch, but the loop jumps back to the expression when done. Thus, the loop's statements may execute multiple times before execution proceeds past the loop.
2. This program receives an input value. If the value > -1, the program adds the value to a sum, receives another input, and repeats. val is 2, so the loop's statements execute, making sum 2.
3. The loop's statements ended by receiving 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 receive 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 content:

Static Figure:
A diagram with an arrow pointing down. There is a defined diamond near the top of the arrow. A series of lines create a loop starting at the diamond and to the right of the downward pointing arrow. This loop is fairly rectangular, but ends with an arrow pointing back to the diamond.
A program is superimposed on the loop and arrow.
The following code appears at the top of the downward pointing arrow, above the diamond:
sum = 0
num = 0
val = Get next input
The following code is displayed next to the diamond and the right horizontal line of the loop:
While val > -1
The following code is displayed inside the loop:
sum = sum + val
num = num + 1
val = Get next input
The following code is displayed between the diamond and the bottom or point of the downward pointing arrow:
avg = sum / num

Put avg to output
Next to the diagram, the text Input is displayed, with a box below that contains the numbers: 2 4 9 -1
The output console contains 1 line of output:
5
Next to the diagram, the following text is displayed:
sum: 0 2 6 15
The 0, 2, and 6 are faded, while the 15 is a stronger color.
Next to the diagram, the following text is displayed
num: 0 1 2 3
The 0, 1, and 2 are faded, while the 3 is a stronger color.

Step 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.
A diagram with an arrow pointing down. There is a defined diamond near the top of the arrow. A series of lines create a loop starting at the diamond and to the right of the downward pointing arrow. This loop is fairly rectangular, but ends with an arrow pointing back to the diamond.
A program appears superimposed on the loop and arrow. Next to the diagram, the text Input appears, with a box below that contains the numbers: 2 4 9 -1
The text Output appears, with an empty box below. This is the output console.
In the diagram, the following code appears at the top of the downward pointing arrow, above the diamond:
sum = 0
num = 0
val = Get next input
The square starts travelling again. The square travels by sum = 0, and the text sum: 0 appears next to the diagram.
The square travels by num = 0, and the text num: 0 appears next to the diagram.
The square travels by val = Get next input, where the value 2 is shown.
The square travels to the diamond, where the text
While val > -1
is displayed. This condition is true, so the square travels along the horizontal line of the loop, going right.
Within the loop, the square travels by the code
sum = sum + val
Then, sum: 2
is displayed beside the loop diagram.
Next, the square travels by the code within the loop
num = num + 1
Then, num: 1
is displayed beside the loop diagram.
Then, the square travels by the code
val = Get next input
within the loop, where 4 is displayed as the value for val.
Then, the square completes travelling around the loop, and returns to the diamond.

Step 2: The next input is 4. The loop is entered, so sum becomes 2 + 4 or 6, and num is incremented to 2.
The square starts at the diamond.
The condition
While val > -1
is true, so the square travels along the horizontal line of the loop, going right.
Within the loop, the square travels by the code
sum = sum + val
Then, sum: 6
is displayed beside the loop diagram.
Next, the square travels by the code within the loop
num = num + 1
Then, num: 2
is displayed beside the loop diagram.
Then, the square travels by the code
val = Get next input
within the loop, where 9 is displayed as the value for val.
Then, the square completes travelling around the loop, and returns to the diamond.

Step 3: The next input is 9, so the loop is entered. Sum becomes 6 + 9 or 15, and num is incremented to 3.
The square starts at the diamond.
The condition
While val > -1
is true, so the square travels along the horizontal line of the loop, going right.
Within the loop, the square travels by the code
sum = sum + val
Then, sum: 15
is displayed beside the loop diagram.
Next, the square travels by the code within the loop
num = num + 1
Then, num: 3
is displayed beside the loop diagram.
Then, the square travels by the code
val = Get next input
within the loop, where -1 is displayed as the value for val.
Then, the square completes travelling around the loop, and returns to the diamond.

Step 4: The next input is -1, so the loop is not entered. 15 / 3 or 5 is output.
The condition
While val > -1
is false, so the square does not enter the loop, and instead travels downward down the downward pointing arrow.

The square travels by the code
avg = sum / num
Then, 15 / 3 is displayed next to this code. Then, the result of this expression 5, is displayed.
Then the square travles by the code
Put avg to output
The output console now contains 1 line of output:
5

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 received 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 received
   - 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 received. If that input was negative (unlike the data in the example above), the loop's body would _____.

- be executed
- not be executed

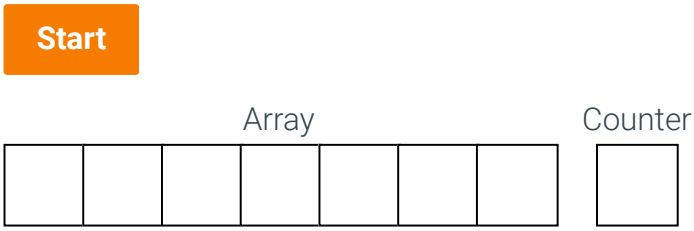
Example: Counting specific values in a list

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.

Start

Array Max

Next value Store value

PARTICIPATION ACTIVITY 5.1.9: Determining the max value.

Complete the program so the variable max ends up 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.

```
curr_power = 2
user_char = 'y'

while user_char == 'y':
    print(curr_power)
    curr_power = curr_power * 2
    user_char = input()

print('Done')
```

Input

y y n

Output

2
4
8
Done

Python interpreter

16	curr_power
'n'	user_char

Animation content:

Static Figure:
Begin Python code:

```
curr_power = 2
user_char = 'y'

while user_char == 'y':
   print(curr_power)
   curr_power = curr_power * 2
   user_char = input()

print('Done')
```

End Python code.

The input console contains 1 line of text:
y y n

The output console contains four lines of output:
2
4
8
Done

The Python interpreter contains two boxes.
The first box has a value of 16, with a label of curr_power
The second box has a value of 'n', with a label of user_char

Step 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.
The input console contains 1 line of text:
y y n
The output console is empty.
A column of boxes titled Python Interpreter is displayed. The boxes are initially empty.
The line of code
curr_power = 2
is highlighted.
The first box of the python interpreter now contains 2, with the label curr_power
The line of code
user_char = 'y'
is highlighted.
The second box of the python interpreter now contains 'y', with the label user_char
The line of code
while user_char == 'y':
is highlighted.

Step 2: Thus, the loop body is executed, which outputs curr_power's current value of 2, doubles curr_power, and gets the next input.

The line of code
```
print(curr_power)
```
is highlighted.
The output console now contains one line of output:
2
The line of code
```
curr_power = curr_power * 2
```
is highlighted.
The first box of the Python interpreter now has a value of 4, with a label of curr_power.
The line of code
```
user_char = input()
```
is highlighted.
The first letter in the input console, y, is highlighted.

Step 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.
The line of code
```
while user_char == 'y':
```
is highlighted.
The first letter in the input console, y, is also highlighted. This while condition is true.
Next, the lines of code
```
print(curr_power)
curr_power = curr_power * 2
user_char = input()
```
are highlighted.
The output console now contains two lines of output:
2
4
The first box of the Python interpreter now has a value of 8, with a label of curr_power.
The second letter in the input console, y, is highlighted.

Step 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.
The line of code
```
while user_char == 'y':
```
is highlighted.
The second letter in the input console, y, is also highlighted. This while condition is true.
Next, the lines of code
```
print(curr_power)
curr_power = curr_power * 2
user_char = input()
```
are highlighted.
The output console now contains three lines of output:
2

4
8
The first box of the Python interpreter now has a value of 16, with a label of curr_power.
The third letter in the input console, n, is highlighted.
The second box of the Python interpreter now has a value of 'n', with a label of user_char.

Step 5: user_char is now 'n', so user_char == 'y' is false. Thus, execution jumps to after the loop, which outputs "Done".

The line of code
while user_char == 'y':
is highlighted.
The third letter in the input console, n, is also highlighted. This while condition is false.
The line of code
print('Done')
is highlighted.
The output console now contains four lines of output:
2
4
8
Done

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.

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

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

Check Show answer

2) Assume the 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 five times. In this case, the value of user_value may be "-", so the result of the multiplication is "-----". Another valid (but long) method is the statement
`print(f'{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

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

# Loop until user enters sentinel value
while user_value != 'q':
    print(f' {user_value} {user_value} ')  # Print eyes
    print(f'  {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): ")
    user_value = user_input[0]

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

```
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.

```
...
# Read num from user ...

# Print each digit
while num > 0:
    print(num % 10)
    num = num // 10
```

Iteration	num	Output
	902	
1		2
	90	
2		0
	9	
3		9
	0	

Printing out every digit of a user-entered number using the % and // operators.

Animation content:

Static Figure:
Begin Python code:

```
...
# Read num from user ...

# Print each digit
while num > 0:
   print(num % 10)
   num = num // 10
```

End Python code.

The results of all iterations is shown in a table:

Iteration	num	Output
	902	
1		2
	90	
2		0
	9	
3		9
	0	

Step 1: User enters the number 902. The first iteration prints "2".
An empty table with 3 columns appears. The columns are labelled Iteration, num, and Output.
The line of code

```
# Read num from user ...
```

is highlighted.
The number 902 appears, and is placed into the info table as the value for num.
The line of code

```
while num > 0:
```

is highlighted.

The number 1 is placed into the table as the value for Iteration.
The line of code
```
    print(num % 10)
```
is highlighted.
The number 2 is placed into the table as the value for the Output.
The line of code
```
    num = num // 10
```
is highlighted.
The number 90 is placed into the table as the value for num.

The current state of the table is:

Iteration	num	Output
	902	
1		2
	90	

Step 2: The second iteration prints "0".
The line of code
```
while num > 0:
```
is highlighted.
The number 2 is placed into the table as the value for Iteration.
The line of code
```
    print(num % 10)
```
is highlighted.
The number 0 is placed into the table as the value for the Output.
The line of code
```
    num = num // 10
```
is highlighted.
The number 9 is placed into the table as the value for num.

The current state of the table is:

Iteration	num	Output
	902	
1		2
	90	
2		0

	9	

Step 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.

The line of code

```
while num > 0:
```

is highlighted.

The number 3 is placed into the table as the value for Iteration.

The line of code

```
    print(num % 10)
```

is highlighted.

The number 9 is placed into the table as the value for the Output.

The line of code

```
    num = num // 10
```

is highlighted.

The number 0 is placed into the table as the value for num.

The current state of the table is:

Iteration	num	Output
	902	
1		2
	90	
2		0
	9	
3		9
	0	

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 largely 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 template...

```
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(f'Ancestors in {year_considered}: {num_ancestors}'

    num_ancestors = num_ancestors * 2
    year_considered = year_considered - years_per_generatior

```

```
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.

566436.4601014.qx3zqy7

Start

Type the program's output

```
g = 0

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

0
1

1 2 3 4

Check Next

CHALLENGE ACTIVITY 5.2.2: While loops.

566436.4601014.qx3zqy7

Start

A while loop reads floating-point numbers from input into variable value_in. Write an expression that ex
value_in is less than -10.0.

▶ **Click here for example**

Note: First, delete the comment `''' Your code goes here '''`. Then, insert your expression for

```
value_in = float(input())
while ''' Your code goes here ''':
    print(f'Value is {value_in}')
    value_in = float(input())

print('Exit')
```


1 2

Check Next level

5.3 More while examples

Greatest common divisor

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 GCD of two positive integers.

Load default template...

```
1  num_a = int(input('Enter fir
2  print()
3
4  num_b = int(input('Enter sec
5  print()
6
7  while num_a != num_b:
8      if num_a > num_b:
9          num_a = num_a - num_
10     else:
11         num_b = num_b - num_
12
13 print(f'GCD is {num_a}')
14
```

```
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. 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 iteration and before the second?

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

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. Use either single (') or double (") quotes.

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.

zyDE 5.3.2: 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...

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

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 in
    if random_num == 0:
        print('\nPlease explain further.\n')
    elif random_num == 1:
        print(f'\nWhy do you say: "{user_text}"?\n')
```

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

Run

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 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.

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

Load default template...

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

```
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.

566436.4601014.qx3zqy7

Start

Type the program's output

```
curr_value = int(input())

maximum_number = curr_value

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

print(f'Max value: {maximum_number}')
```

Input

```
2
34
12
0
```


Output

1 2

Check Next

CHALLENGE ACTIVITY 5.3.2: Bidding example.

Variable keep_bidding is initially assigned with 'y'. Within the while loop, keep_bidding is updated with the user's next input value. Complete the while loop's expression to terminate the while loop if the user enters 'n'.

Ex: If the input is 'y' 'y' 'n', then the output is:

```
I'll bid $10!
Continue bidding? (y/n) I'll bid $15!
Continue bidding? (y/n) I'll bid $21!
Continue bidding? (y/n)
```

Learn how our autograder works

566436.4601014.qx3zqy7

```
1 import random
2 random.seed(5)
3
4 keep_bidding = 'y'
5 next_bid = 0
6
7 while ''' Your solution goes here ''':
```

```
    next_bid = next_bid + random.randint(1, 10)
    print(f'I\'ll bid ${next_bid}!')
    print('Continue bidding? (y/n)', end=' ')
    keep_bidding = input()
```


Run

View your last submission

CHALLENGE ACTIVITY 5.3.3: While loop: Insect growth.

Positive integer num_insects is read from input. Write a while loop that in each iteration first prints and then doubles num_insects. The loop executes while values are less than or equal to 100. Output each number on a newline.

Ex: If the input is: 8, then the output is:

```
8
16
32
64
```

Learn how our autograder works

566436.4601014.qx3zqy7

```
num_insects = int(input())

''' Your solution goes here '''

```

Run

View your last submission

CHALLENGE ACTIVITY | 5.3.4: Advanced while loop examples.

566436.4601014.qx3zqy7

Start

String input_word is read from input. Write a loop that iterates while input_word is *not* equal to 'Tomato' loop:

- Output 'Not Tomato'.
- Read string input_word from input.

▶ **Click here for example**

```
input_word = input()

while ''' Your code goes here ''':

    ''' Your code goes here '''

print('Got Tomato!')
```

1 2

Check Next level

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 $10000 initial savings and 5% yearly interest:

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

Load default template...

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

initial_savings = 10000
interest_rate = 0.05

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

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

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



```
10
```

Run

PARTICIPATION ACTIVITY 5.4.1: Savings interest program.

Refer to the program above.

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

Check Show answer

2) If the interest_rate is 3% and initial_savings are $5000, savings will be greater than $7500 after how many loop iterations? Note: The beginning of year 10 is after 9 loop iterations.

Check Show answer

PARTICIPATION ACTIVITY 5.4.2: Basic counting with while loops.

Use <= in each answer.

1) The loop iterates 10 times.

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

Check Show answer

2) The loop iterates 99 times.

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

Check Show answer

3) The 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 from 5 to 1. The loop body executes when i is 5, 4, 3, 2, and 1, but the loop body 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 the loop body 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 avoid an infinite loop.

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 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


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

Run code

Output is: `Awaiting your input...`

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)*(N-2)*...*2*1). Hint:Use a loop variable i that counts from total-1 down to 1. Compare your 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 use
2  total = N
3  # Initialize the loop variab
4
5  while i ? ??:
6      # Set total to total * (
7      # Decrement i
8
9  print(total)
10
```

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: Loops with variables that count.

566436.4601014.qx3zqy7

Start

Type the program's output

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

1 2 3

Check Next

CHALLENGE ACTIVITY 5.4.2: Counting.

566436.4601014.qx3zqy7

Start

Integer user_num is read from input. Integer j is initialized with 30. Complete the while loop to perform iteration until j is less than user_num:

- Decrement j.
- If j is divisible by 2, then output the value of j.

▶ **Click here for example**

Note: j % 2 == 0 returns True if j is divisible by 2.

```
1 user_num = int(input())
2 j = 30
3
4 while j >= user_num:
5
6     ''' Your code goes here '''
7
```


1 2

Check Next level

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

5.5 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 and takes just 3-5 minutes. Please take a short moment to answer the student survey.

Basics

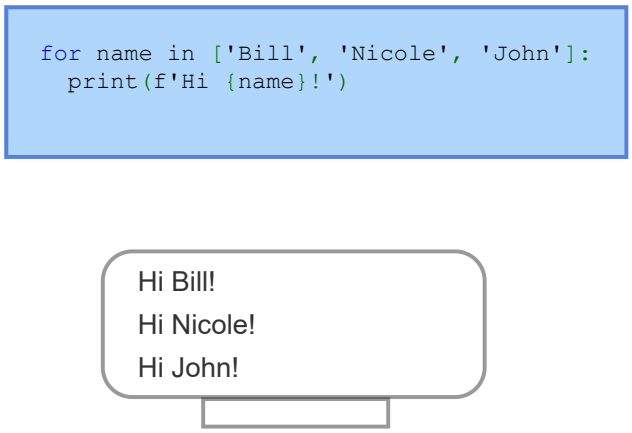
A common programming task is to access all of the elements in a container. Ex: Printing every item in a list. A ***for loop*** statement loops over each element in a container one at a time, assigning a variable with the next element that can then be used in the loop body. The container in the for loop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

Static figure: A code block, output console, and the variable name are displayed.
Begin Python code:
for name in ['Bill', 'Nicole', 'John']:
  print(f'Hi {name}!')
End Python code.
Step 1: The first iteration assigns the variable name with 'Bill' and prints 'Hi Bill!' to the screen. The line of code, print(f'Hi {name}!'), is highlighted and name now contains the text, 'Bill'. The text, Hi Bill!, is output to the console.
Step 2: The second iteration assigns the variable name with 'Nicole' and prints 'Hi Nicole!'. The line of code, for name in ['Bill', 'Nicole', 'John']:, is highlighted and name now contains the text, 'Nicole'. The line of code, print(f'Hi {name}!'), is highlighted and the text, Hi Nicole!, is output to the console.
Step 3: The third iteration assigns the variable name with 'John' and prints 'Hi John!'. The line of code, for name in ['Bill', 'Nicole', 'John']:, is highlighted and name now contains the text, 'John'. The line of code, print(f'Hi {name}!'), is highlighted and the text, Hi John!, is output to the console.

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.

Figure 5.5.1: 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(f'{c} is on channel
{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 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 action during each loop iteration. A simple example is printing the value, as above examples demonstrated. The program below uses an additional variable to sum list elements to calculate weekly and 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(f'Weekly revenue: ${total:.2f}')
print(f'Daily average revenue:
${average:.2f}')
```

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


A for loop may also iterate backward 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 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.

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

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


Check Show answer

CHALLENGE ACTIVITY 5.5.1: Looping over strings, lists, and dictionaries.

566436.4601014.qx3zqy7

Start

Type the program's output


```
numbers = [3, 7, 6]
for number in numbers:
    print(number)
```

1 2 3 4

Check Next

CHALLENGE ACTIVITY 5.5.2: For loops.

566436.4601014.qx3zqy7

Start

Integer num_in is read from input representing the number of values to be read next. The remaining va into paint_list. For each value in paint_list, output the value followed by ' picked' on the same line.

▶ **Click here for example**

```
num_in = int(input())
paint_list = []
for i in range(num_in):
    paint_list.append(input())

print(f'List has {num_in} elements:')

''' Your code goes here '''
```


1 2

Check Next level

5.6 Counting using the range() function

The range() function

While loops are used for counting a specific number of iterations, and for loops are 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 >= 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 >= 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 <= X and > Y, incrementing by Z.
  Ex: `range(3, -1, -1)` creates the sequence 3, 2, 1, 0.

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 to 1
`range(5, 0, -2)`	`5 3 1`	Every 2nd integer from 5 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 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 range() 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 s

initial_savings = 10000
interest_rate = 0.05

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

savings = initial_savings
for i in range(years):
    print(f' Savings in year
    savings = savings + (sav

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.

566436.4601014.qx3zqy7

Start

Type the program's output

```
for i in range(5):
    print(i, end=' ')
```

1 2

Check Next

CHALLENGE ACTIVITY 5.6.2: Counting using the range() function.

566436.4601014.qx3zqy7

Start

Positive integer minutes_up is read from input, representing the number of minutes lapsed since a rec range() function call so that the for loop iterates through the increasing sequence of all non-negative in minutes_up.

▶ **Click here for example**

```
1 minutes_up = int(input())
2
3 print('Minutes recorded:', end=' ')
4
5 for i in range(''' Your code goes here '''):
6     print(i, end=' ')
7 print()
```


1 2 3

Check Next level

5.7 While vs. for loops

While loop and for loop correspondence

Both while loops and for 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.

```
i = 0
while i < 100 :
   # Loop body statements
   i += 1
```

```
for i in range( 100 ):
    # Loop body statements
```

```
i = 0
while   i < 100   :
   #Loop body statements
   i += 1
```

```
for   i   in range( 100 ):
   #Loop body statements
```

Animation content:

Static Figure: Two loop examples, one while loop, one for loop. Begin Python code:
i = 0
while i < 100 :
  # Loop body statements
  i += 1

```
for i in range( 100 ):
  # Loop body statements
```

End Python code.

Step 1: A for loop and range function can replace a while loop. Begin Python code:

```
i = 0
while i < 100 :
  i += 1
```

End Python code. This while loop can be re-written as a for loop. Begin Python code: for i in range(0, 100, 1): End Python code.

Step 2: A more concise for loop uses a single argument for range(). The for loop can be rewritten to be more concise. Begin Python code: for i in range(100): End Python code.

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, but 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 one-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 two-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(f'{letter1}{letter2}.com')
        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 .com domain names.

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

Load default template... Run


```
1 '''
2 Program to print all 2-let
3 Note that ord() and chr()
4 ord('a') is 97. ord('b') i
5 '''
6 print('Two-letter domain r
7
8 letter1 = 'a'
```

```
letter2 = '?'
while letter1 <= 'z':  # (
    letter2 = 'a'
    while letter2 <= 'z':
        print(f'{letter1}{
        letter2 = chr(ord(
    letter1 = chr(ord(lett
```


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 per 5 units. So if the user enters 40, print 8 asterisks.

Load default template...

```
num = 0
while num >= 0:
    num = int(input('Enter a
    if num >= 0:
        print('Depicted grap
        for i in range(num):
            print('*', end='
        print('\n')
print('Goodbye.')
```

```
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(f'{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(f'{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(f'{i1}{i2}',
end=' ')
        i2 = i2 + 3
    i1 = i1 + 10
```

Check Show answer

CHALLENGE ACTIVITY 5.8.1: Enter the output of the nested loop.

566436.4601014.qx3zqy7

Start

Type the program's output

```
count = 0
for i in range(3):
   for j in range(4):
      count = count + 1
print(count)
```

12

1 2 3 4

Check Next

CHALLENGE ACTIVITY 5.8.2: Nested loops: Print rectangle

Given num_rows and num_cols representing the number of rows and columns, write nested loops using range() to print a rectangle as shown in the example below.

Ex: If the input is `2 3`, then the output is:

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

Note: The inner loop should be indented once to contain the inner print statement.

Learn how our autograder works

566436.4601014.qx3zqy7

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

Run

View your last submission

CHALLENGE ACTIVITY 5.8.3: Nested loops: Print seats.

Integers num_rows and num_columns are read from input representing the number of rows and columns of a theater's seating plan. Complete the nested for loop to output each seat label, as shown in the example. Each seat label is followed by a space, and each row is followed by a newline.

Click each step for instructions:

▶ **Step 1**

▶ **Step 2**

▶ **Step 3**

Ex: If the input is:

```
2
3
```

then the output is:

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

Learn how our autograder works

566436.4601014.qx3zqy7

```
num_rows = int(input())
num_columns = int(input())

''' Your solution goes here '''

        print(f'{current_row}{current_column_letter}', end=' ')
        current_column_letter = chr(ord(current_column_letter) + 1)  # Used to increm
    print()
```

Run

View your last submission

CHALLENGE ACTIVITY 5.8.4: Nested loops.

566436.4601014.qx3zqy7

Start

Integers first_num and second_num are read from input. The outer while loop executes first_num times. Write the inner loop to execute (second_num + 1) times for each iteration of the outer while loop.

Ex: If the input is:

```
5
4
```

then the output is:

```
Inner loop ran 25 times
```

```
first_num = int(input())
second_num = int(input())

count = 0
i = 0
while i < first_num:
    j = 0
    while ''' Your code goes here ''':
        count += 1
        j += 1
    i += 1

print(f'Inner loop ran {count} times')
```


1 2 3

Check Next level

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 the program does not work on the first try, debugging can be extra difficult because the program may have many bugs.

Experienced programmers practice ***incremental programming*** by starting with a simple version of the program, and 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 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(f'Element {index} is:
{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(f'Numbers only: {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. The third version adds the elif branch for the letters A-C (lowercase and uppercase, per a standard phone keypad).

Figure 5.9.3: Third version echoes hyphens and handles the 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(f'Numbers only: {phone_number}')
```

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

Load default template...

```
1 user_input = input('Enter ph
```

1-555-HOLIDAY

```
phone_number = ''

for character in user_input:
    if ('0' <= character <=
        phone_number += char
    elif ('a' <= character <
        phone_number += '2'
    #FIXME: Add remaining el
    else:
        phone_number += '?'

print(f'Numbers only: {phone
```

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, a programmer can 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, making the program 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(f'${meal_cost} buys {num_empanadas} empanadas and {num_tacos}
tacos without change.')
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 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
```


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 (Ex: `(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(f'${meal_cost} buys {num_empanadas} empanadas and
{num_tacos} tacos without change.')
            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 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: Enter the output of break and continue.

566436.4601014.qx3zqy7

Start

Type the program's output

Input

5

```
stop = int(input())

for n in range(2, 7):
    print(n)

    if n >= stop:
        print('stop')
        break
```

Output

```
2
3
4
5
stop
```

1 2 3 4

Check Next

CHALLENGE ACTIVITY 5.10.2: 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 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
```

Learn how our autograder works

566436.4601014.qx3zqy7

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

Run

View your last submission

5.11 Loop else

Loop else construct

A loop may include an else clause that executes only if the loop terminates normally and doesn't use 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 infants. 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", pronounced "Albin".)

The program below checks if a user-entered name is an appropriate Danish name. If the name is not found in the list of legal names, then a suggestion is made to a close match. A close match is an acceptable name starting with the same letter. If no close matches are found, the loop else clause informs the user. If there are multiple names with the same letter, the first in the list is used.

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 since there is an acceptable name starting with 'M'.
3. Try the name "Zoidberg", which is not an acceptable name. The list doesn't contain a name starting with 'Z', so the program will print a special message and terminate.

main.py Load default template...

```
#A few legal, acceptable Danish names
legal_names = ['Thor', 'Bjork', 'Bailey', 'Anders', 'Bent
    'Claus', 'Emil', 'Finn', 'Jakob', 'Karen', 'Julie', '
    'Bente', 'Eva', 'Helene', 'Ida', 'Inge', 'Susanne', '
    'Torben', 'Soren', 'Rune', 'Rasmus', 'Per', 'Michael',
    'Dorte'
]

user_name = input('Enter desired name:\n')
if user_name in legal_names:
    print(f'{user_name} is an acceptable Danish name. Cong
else:
    print(f'{user_name} is not acceptable.')
    for name in legal_names:
        if user_name[0] == name[0]:
            print(f'You might consider: {name},', end=' ')
            break
```


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

CHALLENGE ACTIVITY 5.11.1: Loop else.

566436.4601014.qx3zqy7

Start

Type the program's output


```
result = 0
n = 2
while n > -3:
    print(n, end=' ')
    result += 2
    n -= 1
else:
    print(f'\ {result}')
print('done')
```

```
2 1 0 -1 -2 \ 10
done
```

1 2 3

Check Next

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(f'Element {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(f'Element {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(f'Element {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 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

CHALLENGE ACTIVITY 5.12.1: Using enumerate in for loops.

566436.4601014.qx3zqy7

Start

Type the program's output

```
seasons = ['winter', 'spring', 'summer', 'fall']
for (position, season) in enumerate(seasons):
    print(f'{position} {season}')
```

```
0 winter
1 spring
2 summer
3 fall
```

1 2

Check Next

CHALLENGE ACTIVITY 5.12.2: Getting both index and value when looping: enumerate().

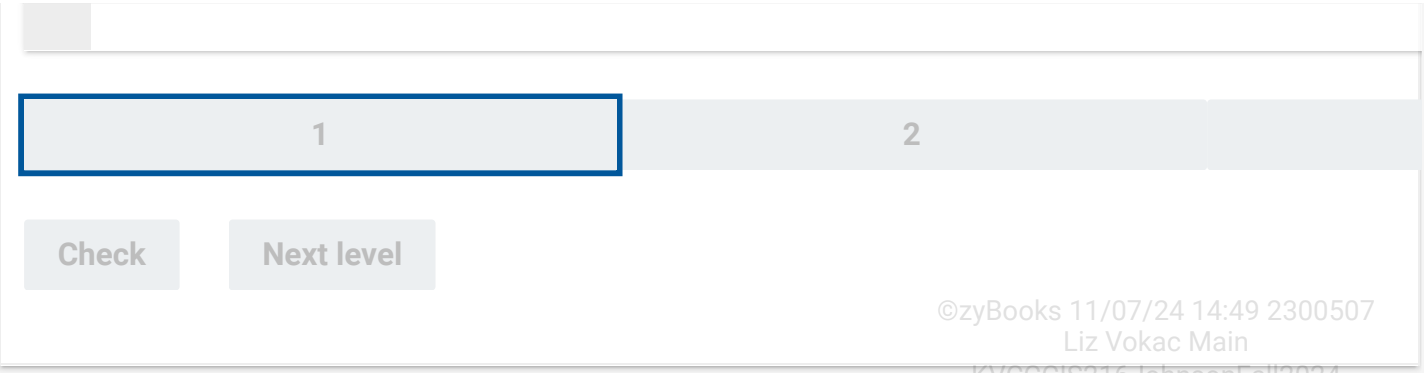
566436.4601014.qx3zqy7

Start

List athlete_positions contains three athlete names read from input. Complete the for loop using the er output each athlete's order on the roster followed by the athlete's name.

▶ **Click here for example**

```
1 athlete_positions = [input(), input(), input()]
2
3 for ''' Your code goes here ''' in enumerate(athlete_positions):
4     print(f'#{k + 1}: {athlete_name}')
```



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. Students planning to fully complete this program may consider first developing their code in a separate programming environment.

Analyzing dice rolls is a frequent 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...

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

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 or equal to 1.
3. prints a histogram in which the total number of times the dice rolls equals each possible value and is displayed by printing a character, such as *, that number of times. Ex:


```
Dice roll histogram:

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

5.14 LAB: Convert to reverse 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 reverse 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 reverse binary | 10 / 10

main.py — Load default template...

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

```

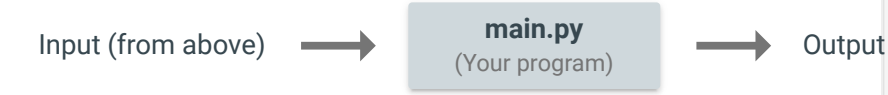

Develop mode | Submit mode

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



Program output displayed here

Coding trail of your work What is this?

```
10/4 F0,0,10
```


5.15 LAB: Password modifier

This section's content is not available for print.

5.16 LAB: Output range with increment of 5

This section's content is not available for print.

5.17 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 "Done", "done", or "d" for the line of text.

Ex: If the input is:

```
Hello there
Hey
done
```

then the output is:

```
ereht olleH
yeH
```


LAB ACTIVITY 5.17.1: LAB: Print string in reverse 2 / 10

main.py


```
1 Loading latest submission...
```