Study Guide: while Loops

This study guide provides a quick-reference summary of what you learned in this segment and serves as a guide for the upcoming practice quiz.

In the **while** Loops segment, you learned about the logical structure and syntax of **while** loops. You also learned about the importance of initializing variables and how to resolve infinite **while** loops with the **break** keyword.

while Loops

A **while** loop executes the body of the loop while a specified condition remains True. They are commonly used when there's an unknown number of operations to be performed, and a condition needs to be checked at each iteration.

Syntax:

```
while specified condition is True:
    body of loop
```

Example **while** loop:

```
multiplier = 1
result = multiplier*5
while result <= 50:
  print(result)
  multiplier += 1
  result = multiplier*5
print("Done")

# This while loop prints the multiples of 5 between 1 and 50. The
# "multiplier" variable is initialized with the starting value of 1.
# The "result" variable is initialized with the value of the
# "multiplier" variable times 5.

# The while loop specifies that the loop must iterate while it is True
# that the "result" is less than or equal to 50. Within the while loop,
# the code tells the Python interpreter to print the value of the
# "result" variable. Then, the "multiplier" is incremented by 1 and the
# "result" is assigned the new value of the "multiplier" times 5.

# The end of the while loop is indicated by the indentation of the next
# line of code moving one tab to the left. At this point, the Python
# interpreter automatically loops back to the beginning of the while
# loop to check the condition again with the new value of the "result"
# variable. When the while loop condition becomes False (meaning
# "result" is no longer less than or equal to 50), the interpreter exits
# the loop and reads the next line of code outside of the loop. In this
# case, that next line tells the interpreter to print the string "Done".

# Click the "Run" button to check the result of this while loop.
```

Run

Reset

Common Errors in while Loops

If you get an error message on a loop or it appears to hang, your debugging checklist should include the following checks:

- **Failure to initialize variables.** Make sure all the variables used in the loop's condition are initialized before the loop.
- **Unintended infinite loops.** Make sure that the body of the loop modifies the variables used in the condition, so that the loop will eventually end for all possible values of the variables. You can often prevent an infinite loop by using the **break** keyword or by adding end criteria to the condition part of the **while** loop.

while Loop Terms

- **while loop** - Tells the computer to execute a set of instructions while a specified condition is True. In other words, **while** loops keep executing the same group of instructions until the condition becomes False.
- **infinite loop** - Missing a method for exiting the loop, causing the loop to run forever.
- **break** - A keyword that can be used to end a loop at a specific point.

Math Concepts on the Practice Quiz

The coding problems on the upcoming practice quiz will involve a few math concepts. Don't worry if you are rusty on math. You will have plenty of support with these concepts on the quiz. The following is a quick overview of the math terms you will encounter on the quiz:

- **prime numbers** - Integers that have only two factors, which are the number itself multiplied by 1. The lowest prime number is 2.
- **prime factors** - Prime numbers that are factors of an integer. For example, the prime numbers 2 and 5 are the prime factors of the number 10 (2x5=10). The prime factors of an integer will not produce a remainder when used to divide that integer.
- **divisor** - A number (denominator) that is used to divide another number (numerator). For example, if the number 10 is divided by 5, the number 5 is the divisor.
- **sum of all divisors of a number** - The result of adding all of the divisors of a number together.
- **multiplication table** - An integer multiplied by a series of numbers and their results formatted as a table or a list. For example:

  4x1=4
  4x2=8
  4x3=12
  4x4=16
  4x5=20

Coding skills

Skill Group 1

- Initialize a variable
- Use a **while** loop that runs while a specific condition is true
- Ensure the **while** loop will not be an infinite loop
- Increment a value within a **while** loop

```
# This function counts the number of integer factors for a
# "given_number" variable, passed through the function's parameters.
# The "count" return value includes the "given_number" itself as a
# factor (n*1).
def count_factors(given_number):

  # To include the "given_number" variable as a "factor", initialize
  # the "factor" variable with the value 1 (if the "factor" variable
  # were to start at 2, the "given_number" itself would be excluded).
  factor = 1
  count = 1

  # This "if" block will run if the "given_number" equals 0.
  if given_number == 0:
    # If True, the return value will be 0 factors.
    return 0

  # The while loop will run while the "factor" is still less than
  # the "given_number" variable.
  while factor < given_number:
    # This "if" block checks if the "given_number" can be divided by
    # the "factor" variable without leaving a remainder. The modulo
    # operator % is used to test for a remainder.
    if given_number % factor == 0:
      # If True, then the "factor" variable is added to the count of
      # the "given_number"'s integer factors.
      count += 1
    # When exiting the if block, increment the "factor" variable by 1
    # to divide the "given_number" variable by a new "factor" value
    # inside the while loop.
    factor += 1

  # When the interpreter exits either the while loop or the top if
  # block, it will return the value of the "count" variable.
  return count

print(count_factors(0)) # Count value will be 0
print(count_factors(3)) # Should count 2 factors (1x3)
print(count_factors(10)) # Should count 4 factors (1x10, 2x5)
print(count_factors(24)) # Should count 8 factors (1x24, 2x12, 3x8,
```

Run

Reset

Skill Group 2

- Initialize variables to assign data types before they are used in a **while** loop
- Use the **break** keyword as an exit point for a **while** loop

```
# This function outputs an addition table. It is written to end after
# printing 5 lines of the addition table, but it will break out of the
# loop if the "my_sum" variable exceeds 20.

# The function accepts a "given_number" variable through its
# parameters.
def addition_table(given_number):

    # The "iterated_number" and "my_sum" variables are initialized with
    # the value of 1. Although the "my_sum" variable does not need any
    # specific initial value, it still must be assigned a data type
    # before being used in the while loop. By initializing "my_sum"
    # with any integer, the data type will be set to int.
    iterated_number = 1
    my_sum = 1

    # The while loop will run while it is True that the
    # "iterated_number" is less than or equal to 5.
    while iterated_number <= 5:

        # The "my_sum" variable is assigned the value of the
        # "given_number" plus the "iterated_number" variables.
        my_sum = given_number + iterated_number

        # Test to see if the "my_sum" variable is greater than 20.
        if my_sum > 20:
            # If True, then use the break keyword to exit the loop.
            break
        # If False, the Python interpreter will move to the next line
        # in the while loop after the if-statement has ended.

        # The print function will output the "given_number" plus
```