Congratulations! You passed!

Grade received 100% To pass 80% or higher

Go to next item

1. **In Python,** what do **while** loops do? 1 / 1 point

- (●) while loops tell the computer to repeatedly execute a set of instructions while a condition is true.
- ( ) while loops instruct the computer to execute a piece of code a set number of times.
- ( ) while loops branch execution based on whether or not a condition is true.
- ( ) while loops initialize variables in Python.

Correct
Correct. while loops will keep executing the same group of instructions until a specified loop condition stops being true.

2. Which techniques can prevent an infinite while loop? Select all that apply. 1 / 1 point

- [x] Change the value of a variable used in the while condition

Correct
Correct. If a numeric variable is used to control the iterations of a while condition, that variable must eventually change to a value that makes the while condition False, otherwise the while loop will keep executing indefinitely.

- [ ] Use the **stop** keyword
- [x] Use the **break** keyword

Correct
Correct. The **break** keyword can stop a while loop. It is often used in the body of a nested if-else conditional statement inside of a while loop.

- [ ] Use the **continue** keyword

3. The following code contains an infinite loop, meaning the Python interpreter does not know when to exit the loop once the task is complete. To solve the problem, you will need to: 1 / 1 point

1. Find the error in the code
2. Fix the **while** loop so there is an exit condition

Hint: Try running your function with the number 0 as the input and observe the result.

Note that Coursera's code blocks will time out after 5 seconds of running an infinite loop. If you get this timeout error message, it means the infinite loop has not been fixed.

```
1  def is_power_of_two(number):
2    # This while loop checks if the "number" can be divided by two
3    # without leaving a remainder. How can you change the while loop to
4    # avoid a Python ZeroDivisionError?
5    while number != 0 and number % 2 == 0:
6      number = number / 2
7    # If after dividing by 2 "number" equals 1, then "number" is a power
8    # of 2.
9    if number == 1:
```

```
    return True
  return False


# Calls to the function
print(is_power_of_two(0)) # Should be False
print(is_power_of_two(1)) # Should be True
print(is_power_of_two(8)) # Should be True
print(is_power_of_two(9)) # Should be False
```

Run

Reset

Correct

Correct. You fixed a tricky error that was hard to find and the function now behaves correctly.

4. Fill in the blanks to complete the while loop so that it returns the sum of all the divisors of a number, without including the number itself. As a reminder, a divisor is a number that divides into another without a remainder. To do this, you will need to:

1 / 1 point

1. Initialize the "divisor" and "total" variables with starting values
2. Complete the while loop condition
3. Increment the "divisor" variable inside the while loop
4. Complete the return statement

```
# Fill in the blanks so that the while loop continues to run while the
# "divisor" variable is less than the "number" parameter.

def sum_divisors(number):
# Initialize the appropriate variables
  divisor = 1
  total = 0

  # Avoid dividing by 0 and negative numbers
  # in the while loop by exiting the function
  # if "number" is less than one
  if number < 1:
    return 0

  # Complete the while loop
  while divisor < number :
    if number % divisor == 0:
      total += divisor
    # Increment the correct variable
    divisor += 1

  # Return the correct variable
  return total


print(sum_divisors(0)) # Should print 0
print(sum_divisors(3)) # Should print 1
# 1
print(sum_divisors(36)) # Should print 1+2+3+4+6+9+12+18
# 55
print(sum_divisors(102)) # Should print 1+2+3+6+17+34+51
# 114

```

Run

Reset

Correct

Correct. You've written a complex while loop and got Python to do the work for you.

5. Fill in the blanks to complete the function, which should output a multiplication table. The function accepts a variable “number” through its parameters. This “number” variable should be multiplied by the numbers 1 through 5, and printed in a format similar to “1x6=6” (“number” x “multiplier” = “result”). The code should also limit the “result” to not exceed 25. To satisfy these conditions, you will need to:

1 / 1 point

1. Initialize the “multiplier" variable with the starting value
2. Complete the while loop condition
3. Add an exit point for the loop
4. Increment the “multiplier" variable inside the while loop

```
def multiplication_table(number):
    # Initialize the appropriate variable
    multiplier = 1

    # Complete the while loop condition.
    while multiplier <= 5 :
        result = number * multiplier
        if  result > 25 :
            # Enter the action to take if the result is greater than 25
            break
        print(str(number) + "x" + str(multiplier) + "=" + str(result))

        # Increment the appropriate variable
        multiplier += 1


multiplication_table(3)
# Should print:
# 3x1=3
# 3x2=6
# 3x3=9
# 3x4=12
# 3x5=15

multiplication_table(5)
# Should print:
# 5x1=5
# 5x2=10
# 5x3=15
# 5x4=20
# 5x5=25

multiplication_table(8)
# Should print:
# 8x1=8
# 8x2=16
# 8x3=24

```

Run

Reset

Correct

Correct. You completed the multiplication table with all of the required criteria, and it looks great!