Week 3 Quiz

**Practice Quiz: While Loops**

Question 1

What are while loops in Python?



**While loops let the computer 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 let us branch execution on whether or not a condition is true.



While loops are how we initialize variables in Python.

**Correct**

Right on! Using while loops we can keep executing the same group of instructions until the condition stops being true.

**1 / 1 point**

Question 2

Fill in the blanks to make the print\_prime\_factors function print all the prime factors of a number. A prime factor is a number that is prime and divides another without a remainder.



**def print\_prime\_factors(number):**

**# Start with two, which is the first prime**

**factor = 2**

**# Keep going until the factor is larger than the number**

**while factor <= number:**

**# Check if factor is a divisor of number**

**if number % factor == 0:**

**# If it is, print it and divide the original number**

**print(factor)**

**number = number / factor**

**else:**

**# If it's not, increment the factor by one**

**factor=factor+1**

**return "Done"**

**print\_prime\_factors(100)**

**# Should print 2,2,5,5**

**# DO NOT DELETE THIS COMMENT**

**Correct**

You nailed it! You've got the code to print all the right

prime factors. Well done!

**1 / 1 point**

Question 3

The following code can lead to an infinite loop. Fix the code so that it can finish successfully for all numbers.

Note: Try running your function with the number 0 as the input, and see what you get!



**def is\_power\_of\_two(n):**

**# Check if the number can be divided by two without a remainder**

**while n % 2 == 0:**

**if n==0:**

**break**

**else:**

**n=n/2**

**# If after dividing by two the number is 1, it's a power of two**

**if n == 1:**

**return True**

**return False**

**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**

**Correct**

Awesome! You fixed a tricky error that was hard to find and

the function now behaves correctly.

**1 / 1 point**

Question 4

Fill in the empty function so that it returns the sum of all the divisors of a number, without including it. A divisor is a number that divides into another without a remainder.



**def sum\_divisors(n):**

**sum = 0**

**# Return the sum of all divisors of n, not including n**

**for i in range(1,n):**

**if n%i==0:**

**sum=sum+i**

**return sum**

**print(sum\_divisors(0))**

**# 0**

**print(sum\_divisors(3)) # Should sum of 1**

**# 1**

**print(sum\_divisors(36)) # Should sum of 1+2+3+4+6+9+12+18**

**# 55**

**print(sum\_divisors(102)) # Should be sum of 2+3+6+17+34+51**

**# 114**

**Correct**

Well done, you! You've written a complex while loop and got

Python to do the work for you.

**1 / 1 point**

Question 5

The multiplication\_table function prints the results of a number passed to it multiplied by 1 through 5. An additional requirement is that the result is not to exceed 25, which is done with the break statement. Fill in the blanks to complete the function to satisfy these conditions.



**def multiplication\_table(number):**

**# Initialize the starting point of the multiplication table**

**multiplier = 1**

**# Only want to loop through 5**

**while multiplier <= 5:**

**result = number\*multiplier**

**# What is the additional condition to exit out of the loop?**

**if result>25 :**

**break**

**print(str(number) + "x" + str(multiplier) + "=" + str(result))**

**# Increment the variable for the loop**

**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**

**Correct**

Excellent! You completed the multiplication table with all

of the required criteria, and it looks great!

**1 / 1 point**

**Practice Quiz: For Loops**

Question 1

How are while loops and for loops different in Python?



While loops can be used with all data types, for loops can only be used with numbers.



For loops can be nested, but while loops can't.



**While loops iterate while a condition is true, for loops iterate through a sequence of elements.**



While loops can be interrupted using break, for loops using continue.

**Correct**

**You got it! We can use while loops when we want our code to execute repeatedly while a condition is true, and for loops when we want to execute a block of code for each element of a sequence.**

**1 / 1 point**

Question 2

Fill in the blanks to make the factorial function return the factorial of n. Then, print the first 10 factorials (from 0 to 9) with the corresponding number. Remember that the factorial of a number is defined as the product of an integer and all integers before it. For example, the factorial of five (5!) is equal to 1\*2\*3\*4\*5=120. Also recall that the factorial of zero (0!) is equal to 1.



**def factorial(n):**

**result = 1**

**for x in range(1,n+1):**

**result = result \* x**

**return result**

**for n in range(0,10):**

**print(n, factorial(n+0))**

**0 1**

**1 1**

**2 2**

**3 6**

**4 24**

**5 120**

**6 720**

**7 5040**

**8 40320**

**9 362880**

**Correct**

**Great work! The pieces of code you're tackling keep getting**

**more complex, you're doing a great job!**

**1 / 1 point**

Question 3

Write a script that prints the first 10 cube numbers (x\*\*3), starting with x=1 and ending with x=10.



**for x in range(1,11):**

**print(x\*x\*x)**

**1**

**8**

**27**

**64**

**125**

**216**

**343**

**512**

**729**

**1000**

**Correct**

**You nailed it! You got the code to print the first 10 cubes.**

**1 / 1 point**

Question 4

Write a script that prints the multiples of 7 between 0 and 100. Print one multiple per line and avoid printing any numbers that aren't multiples of 7. Remember that 0 is also a multiple of 7.



**for n in range(101):**

**if n%7==0:**

**print(n)**

**0**

**7**

**14**

**21**

**28**

**35**

**42**

**49**

**56**

**63**

**70**

**77**

**84**

**91**

**98**

**Correct**

**Awesome! You're getting Python to do all the work for you.**

**1 / 1 point**

Question 5

The retry function tries to execute an operation that might fail, it retries the operation for a number of attempts. Currently the code will keep executing the function even if it succeeds. Fill in the blank so the code stops trying after the operation succeeded.



**def retry(operation, attempts):**

**for n in range(attempts):**

**if operation():**

**print("Attempt " + str(n) + " succeeded")**

**break**

**else:**

**print("Attempt " + str(n) + " failed")**

**retry(create\_user, 3)**

**retry(stop\_service, 5)**

**RunReset**

**Attempt 0 failed**

**Attempt 1 failed**

**Attempt 2 succeeded**

**Attempt 0 succeeded**

**Attempt 0 failed**

**Attempt 1 failed**

**Attempt 2 failed**

**Attempt 3 succeeded**

**None**

**Correct**

**Well done, you! You've fixed the code to stop executing once**

**the function is successful.**

**1 / 1 point**

**Practice Quiz: Recursion**

Question 1

What is recursion used for?



Recursion is used to create loops in languages where other loops are not available.



We use recursion only to implement mathematical formulas in code.



Recursion is used to iterate through sequences of files and directories.



**Recursion lets us tackle complex problems by reducing the problem to a simpler one.**

**Correct**

**You nailed it! By reducing the problem to a smaller one each time a recursive function is called, we can tackle complex problems in simple steps.**

**1 / 1 point**

Question 2

Which of these activities are good use cases for recursive programs? Check all that apply.



**Going through a file system collecting information related to directories and files.**

**Correct**

**Right on! Because directories can contain subdirectories that can contain more subdirectories, going through these contents is a good use case for a recursive program.**



Creating a user account.



Installing or upgrading software on the computer.



**Managing permissions assigned to groups inside a company, when each group can contain both subgroups and users.**

**Correct**

**You got it! As the groups can contain both groups and users, this is the kind of problem that is a great use case for a recursive solution.**



Checking if a computer is connected to the local network.

**1 / 1 point**

Question 3

Fill in the blanks to make the is\_power\_of function return whether the number is a power of the given base. Note: base is assumed to be a positive number. Tip: for functions that return a boolean value, you can return the result of a comparison.



**def is\_power\_of(number, base):**

**# Base case: when number is smaller than base.**

**if number < base:**

**# If number is equal to 1, it's a power (base\*\*0).**

**return number==1**

**# Recursive case: keep dividing number by base.**

**return is\_power\_of(number/base, base)**

**print(is\_power\_of(8,2)) # Should be True**

**print(is\_power\_of(64,4)) # Should be True**

**print(is\_power\_of(70,10)) # Should be False**

**Correct**

**Nice job! You've made the code check for powers of numbers**

**by reducing the problem to a smaller one.**

**1 / 1 point**

Question 4

The count\_users function recursively counts the amount of users that belong to a group in the company system, by going through each of the members of a group and if one of them is a group, recursively calling the function and counting the members. But it has a bug! Can you spot the problem and fix it?



**def count\_users(group):**

**count = 0**

**for member in get\_members(group):**

**count += 1**

**if is\_group(member):**

**count += count\_users(member)-1**

**return count**

**print(count\_users("sales")) # Should be 3**

**print(count\_users("engineering")) # Should be 8**

**print(count\_users("everyone")) # Should be 18**

**Correct**

**Well done, you! You spotted the problem that was causing**

**groups to be counted when we only wanted to count users!**

**1 / 1 point**

Question 5

Implement the sum\_positive\_numbers function, as a recursive function that returns the sum of all positive numbers between the number n received and 1. For example, when n is 3 it should return 1+2+3=6, and when n is 5 it should return 1+2+3+4+5=15.



**def sum\_positive\_numbers(n):**

**if n<=1:**

**return n**

**else:**

**return n+sum\_positive\_numbers(n-1)**

**print(sum\_positive\_numbers(3)) # Should be 6**

**print(sum\_positive\_numbers(5)) # Should be 15**

**Correct**

**Here is your output:**

**6**

**15**

**Great work! You've really nailed writing recursive**

**functions!**

**1 / 1 point**