Study Guide: Week 2 Graded Quiz

It is time to prepare for the Week 2 graded quiz. Please review the following items from this week before starting the Week 2 Graded Quiz. If you would like to refresh your memory on these materials, please revisit the Study Guides located before each Practice Quiz in Week 2 : Study Guide: Expressions and Variables, Study Guide: Functions, and Study Guide: Conditionals.

Knowledge

- How to assign values to variables and use them in code
- How to construct a function and use function parameters
- How comparison and logical operators can be used,
- How comparison and logical operators behave with different data types
- What type of results simple and complex comparisons produce
- How to alphabetize strings using comparison operators
- What must appear after the **if** and **elif** keywords
- What the **elif** keyword does
- When an **if, elif,** or **else**-statement will execute
- How to use the floor division // and modulo % operators and why
- How to use logical operators with comparison operators to develop complex conditional statements within an **if-elif-else** block
- Best practices for coding and their benefits
- What "self-documenting code" means

There may be a few questions on the quiz that will ask you about *either* the *output* of a small block of code *or the value of part of the code.* Make sure to read the instructions carefully on those questions.

Coding skills

Skill Group 1

- Use a function with the def() keyword
- Pass a parameter to the function
- Use an if-elif-else block to set specific conditions for a variety of actions
- Assign strings to variables
- Use comparison operators
- Return a value
- Call the function in a print statement and pass parameter to the function

```
# A function is created with the def() keyword. The parameter
# variable "time_as_string" is passed to the function through a
# call to the function.
def task_reminder(time_as_string):

    # The following if-elif-else block assigns various strings to
    # the variable "task" depending on specific conditions. The
    # test conditions are set using the == equality comparison
    # operator. In this case, the time passed through the
    # "time_as_string" parameter variable is tested as the
    # specific condition. So, if the time  is "11:30 a.m.", then
    # "task" is assigned the value: "Run TPS report".
    if time_as_string == "8:00 a.m.":
        task = "Check overnight backup images"
    elif time_as_string == "11:30 a.m.":
        task = "Run TPS report"
    elif time_as_string == "5:30 p.m.":
        task = "Reboot servers"
    # The else statement is a catchall for all other values of
    # the "time_as_string" parameter variable not listed in the
    # if-elif block of code.
    else:
        task = "Provide IT Support to employees"

    # This line returns the value of "task" to the function call.
    return task

# This line calls the function and passes a parameter
# ("10:00 a.m.") to the function.
print(task_reminder("10:00 a.m."))
# Should print "Provide IT Support to employees"

```

Run

Reset

Skill Group 2

- Predict the output of expressions written with Python's syntax.
- Requires an understanding of:
  - Arithmetic and logical operators
  - How functions return and print values
  - How if-elif-else statements work
  - Comparison operators

```
# Example 1
# Evaluate the output of this print statement

def product(a, b):
        return(a*b)

print(product(product(2,4), product(3,5)))

###################################

# Example 2
# Evaluate the output of this print statement

def difference(a, b):
        return(a-b)

def sum(a, b):
        return(a+b)

print(difference(sum(2,2), sum(3,3)))


###################################


# Example 3
# Evaluate the Boolean output of this comparison


print((5 >= 2*4) and (5 <= 4*3))


###################################


# Example 4
# Evaluate the value of the comparison in the if statement

x = 3
```

Run

Reset

Skill Group 3

- Create an if-elif-else statement with:
  - a complex conditional statement using both comparison and logical operators
  - values assigned to variables
  - arithmetic operators, including the modulo % operator

```
def get_remainder(x, y):

  if x == 0 or y == 0 or x ==y:
    remainder = 0
  else:
    remainder = (x % y) / y
  return remainder


print(get_remainder(10, 3))

```

Run

Reset

Reminder: Correct syntax is critical

Using precise syntax is critical when writing code in any programming language, including Python. Even a small typo can cause a syntax error and the automated Python-coded quiz grader will mark your code as incorrect. This reflects real life coding errors in the sense that a single error in spelling, case, punctuation, etc. can cause your code to fail. Coding problems caused by imprecise syntax will always be an issue whether you are learning a programming language or you are using programming skills on the job. So, it is critical to start the habit of being precise in your code now.

No credit will be given if there are any coding errors on the automated graded quizzes - including minor errors. Fortunately, you have 3 optional retake opportunities on the graded quizzes in this course. Additionally, you have unlimited retakes on practice quizzes and can review the videos and readings as many times as you need to master the concepts in this course.

Now, before starting the graded quiz, please review this list of common syntax errors coders make when writing code.

Common syntax errors:

- Misspellings
- Incorrect indentations
- Missing or incorrect key characters:
  - Parenthetical types - (curved), [square], { curly }