Practice Quiz: Lists

TOTAL POINTS 6

1. Given a list of filenames, we want to rename all the files with extension hpp to the extension h. To do this, we would like to generate a new list called newfilenames, consisting of the new filenames. Fill in the blanks in the code using any of the methods you've learned thus far, like a for loop or a list comprehension.

1 / 1 point

```python
filenames = ["program.c", "stdio.hpp", "sample.hpp", "a.out", "math.hpp", "hp
  .out"]
# Generate newfilenames as a list containing the new filenames
# using as many lines of code as your chosen method requires.
newfilenames = [file.replace('.hpp','.h') for file in filenames]

print(newfilenames)
# Should be ["program.c", "stdio.h", "sample.h", "a.out", "math.h", "hpp.out"
```

Run

Reset

```
['program.c', 'stdio.h', 'sample.h', 'a.out', 'math.h', 'hpp.out']
```

✓ **Correct**

Great work! You're starting to see the benefits of knowing how to operate with lists and strings.

2. Let's create a function that turns text into pig latin: a simple text transformation that modifies each word moving the first character to the end and appending "ay" to the end. For example, python ends up as ythonpay.

1 / 1 point

```python
def pig_latin(text):
  say = []
  # Separate the text into words
  words = text.split()
  for word in words:
    # Create the pig latin word and add it to the list
    say.append(word[1:] + word[0] + 'ay')
    # Turn the list back into a phrase
  return " ".join(say)

print(pig_latin("hello how are you")) # Should be "ellohay owhay reaay ouyay"
print(pig_latin("programming in python is fun")) # Should be "rogrammingpay i
  ythonpay siay unfay"
```

Run

Reset

```
ellohay owhay reaay ouyay
rogrammingpay niay ythonpay siay unfay
```

✓ **Correct**

Nice! You're using some of the best string and list functions to make this work. Great job!

3. The permissions of a file in a Linux system are split into three sets of three permissions: read, write, and execute for the owner, group, and others. Each of the three values can be expressed as an octal number summing each permission, with 4 corresponding to read, 2 to write, and 1 to execute. Or it can be written with a string using the letters r, w, and x or - when the permission is not granted. For example: 640 is read/write for the owner, read for the group, and no permissions for the others; converted to a string, it would be: "rw-r-----" 755 is read/write/execute for the owner, and read/execute for group and others; converted to a string, it would be: "rwxr-xr-x" Fill in the blanks to make the code convert a permission in octal format into a string format. **1 / 1 point**

```python
def octal_to_string(octal):
    result = ""
    value_letters = [(4,"r"),(2,"w"),(1,"x")]
    # Iterate over each of the digits in octal
    for digit in [int(n) for n in str(octal)]:
        # Check for each of the permissions values
        for value, letter in value_letters:
            if digit >= value:
                result += letter
                digit -= value
            else:
                result += '-'
    return result

print(octal_to_string(755)) # Should be rwxr-xr-x
print(octal_to_string(644)) # Should be rw-r--r--
print(octal_to_string(750)) # Should be rwxr-x---
print(octal_to_string(600)) # Should be rw-------
```

Run

Reset

```
rwxr-xr-x
rw-r--r--
rwxr-x---
rw-------
```

✓ **Correct**

You nailed it! This is how we work with lists of tuples, how exciting is that!

4. Tuples and lists are very similar types of sequences. What is the main thing that makes a tuple different from a list? **1 / 1 point**

- ○ A tuple is mutable
- ○ A tuple contains only numeric characters
- ● A tuple is immutable
- ○ A tuple can contain only one type of data at a time

✓ **Correct**
Awesome! Unlike lists, tuples are immutable, meaning they can't be changed.

5. The group_list function accepts a group name and a list of members, and returns a string with the format: group_name: member1, member2, … For example, group_list("g", ["a","b","c"]) returns "g: a, b, c". Fill in the gaps in this function to do that.

1 / 1 point

```
def group_list(group, users):
  members = users
  return ("{}: {}".format(group,', '.join(members)))

print(group_list("Marketing", ["Mike", "Karen", "Jake", "Tasha"])) # Should be
  "Marketing: Mike, Karen, Jake, Tasha"
print(group_list("Engineering", ["Kim", "Jay", "Tom"])) # Should be "Engineer
  : Kim, Jay, Tom"
print(group_list("Users", "")) # Should be "Users:"
```

Run

Reset

```
Marketing: Mike, Karen, Jake, Tasha
Engineering: Kim, Jay, Tom
Users:
```

✓ **Correct**

Nice job! You're doing well, working with list elements!

6. The guest_list function reads in a list of tuples with the name, age, and profession of each party guest, and prints the sentence "Guest is X years old and works as __." for each one. For example, guest_list(('Ken', 30, "Chef"), ("Pat", 35, 'Lawyer'), ('Amanda', 25, "Engineer")) should print out: Ken is 30 years old and works as Chef. Pat is 35 years old and works as Lawyer. Amanda is 25 years old and works as Engineer. Fill in the gaps in this function to do that.

1 /

```
def guest_list(guests):
  for guest in guests:
    name, age, job = guest
    print("{name} is {age} years old and works as {job}".format(name=name, age
      =age, job=job))

guest_list([('Ken', 30, "Chef"), ("Pat", 35, 'Lawyer'), ('Amanda', 25,
  "Engineer")])
```

Run

Reset

```
Ken is 30 years old and works as Chef
Pat is 35 years old and works as Lawyer
Amanda is 25 years old and works as Engineer
Ken is 30 years old and works as Chef
Pat is 35 years old and works as Lawyer
Amanda is 25 years old and works as Engineer
None
```

✓ **Correct**

Well done! See how the format methodology combines with tuple functionality to easily create interesting code!