Congratulations! You passed!

Grade received 100% **To pass** 80% or higher

Go to next item

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**

```
filenames = ["program.c", "stdio.hpp", "sample.hpp", "a.out", "math.hpp", "hpp.out"]
# Generate newfilenames as a list containing the new filenames
# using as many lines of code as your chosen method requires.
newfilenames = [x.replace("hpp","h") if x.endswith("hpp") else x for x 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**

```
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 += word[1:] + word[0] + "ay" + " "
    # Turn the list back into a phrase
  return 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 niay ythonp
```

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. Which list method can be used to add a new element to a list at a specified index position? **1 / 1 point**

- ○ list.pop(index)
- ◉ list.insert(index, x)
- ○ list.add(index, x)
- ○ list.append(x)

Correct

Correct! The list.insert(index, x) method will insert the given "x" element into the list at the specified index position.

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

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 = ""
  for user in users:
    members = ", ".join(users)
  return ("{}: {}".format(group,members))

print(group_list("Marketing", ["Mike", "Karen", "Jake", "Tasha"])) # Should be "Marketi
print(group_list("Engineering", ["Kim", "Jay", "Tom"])) # Should be "Engineering: Kim,
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 / 1 point

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

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

#Click Run to submit code
"""
Output should match:
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
"""
```

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
```

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