List Comprehension Examples

You can create a list from a sequence using a **for** loop, but there's a more streamlined way to do this by using a list comprehension. List comprehensions allow you to create a new list from a sequence or a range in a single line.

Simple List Comprehension

For example, **[x*2 for x in range(1,11)]** is a simple list comprehension. This single line of code iterates over a range from 1 to 10, multiplies each element in the range by 2, and creates a new list from all multiples of 2 from 2 to 20.

```
### Simple List Comprehension
print("List comprehension result:")

# The following list comprehension compacts several lines
# of code into one line:
print([x*2 for x in range(1,11)])



### Long form for loop
print("Long form code result:")

# The list comprehension above accomplishes the same result as
# the long form version of the code:
my_list = []
for x in range(1,11):
   my_list.append(x*2)
print(my_list)


# Click Run to compare the two results.
```

Run

Reset

List Comprehension with Conditional Statement

You can also use conditionals with list comprehensions to build even more complex and powerful statements. You can do this by appending an if statement to the end of the list comprehension. For example, **[x for x in range(1,101) if x % 10 == 0]** generates a new list containing all the integers divisible by 10 from 1 to 100. The if statement evaluates each value in the range from 1 to 100 to check if it's evenly divisible by 10. If it is, the number is added to a new list.

```
### List Comprehension with Conditional Statement
print("List comprehension result:")

# The following list comprehension compacts multiple lines
# of code into one line:
print([ x for x in range(1,101) if x % 10 == 0 ])

### Long form for loop with nested if-statement
print("Long form code result:")

# The list comprehension above accomplishes the same result as
# the long form version of the code:
my_list = []
for x in range(1,101):
  if x % 10 == 0:
    my_list.append(x)
print(my_list)


# Click Run to observe the two results.
```

Run

Reset

List comprehensions can be really powerful, but they can also be complex, resulting in code that's hard to read. Be careful when using them, since it might make it more difficult for someone else looking at your code to easily understand what the code is doing. It is a best practice to add descriptive comments about any list comprehensions used in your code. This helps to communicate the purpose of list comprehensions to other coders. Comments will also help you remember the goal of the code when performing future code additions and maintenance.

Practice exercise

This exercise will walk you through how to write a list comprehension to create a list of squared numbers (n*n). It needs to return a list of squares of consecutive numbers between "start" and "end" *inclusively*. For example, squares(2, 3) should return a list containing [4, 9].

1. The function receives the variables "start" and "end" through the function parameters.
2. In the **return** line, start by entering the list brackets [ ]
3. Between the brackets [ ], enter the arithmetic expression to square a variable "n".
4. To the right of the square expression, write a **for** loop that iterates over "n" in a range from the "start" to "end" variables.
5. Ensure the "end" range value is included in the **range()** by adding 1 to it.
6. Run your code to see if it works! If needed, the solution to this code is included in the "Study Guide: List Operations and Methods" reading under "Skill Group 2" (list comprehensions).

```
def squares(start, end):
    return ___


print(squares(2, 3))  # Should print [4, 9]
print(squares(1, 5))  # Should print [1, 4, 9, 16, 25]
print(squares(0, 10)) # Should print [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Run

Reset

Mark as completed

Like **Dislike** **Report an issue**