Python: 5th lesson – Loops & List Comprehensions

Loops:

* Loops are a way to repeatedly execute some code. Here's an example:

planets = ['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']

for planet in planets:

print(planet, end=' ') # print all on same line

Mercury Venus Earth Mars Jupiter Saturn Uranus Neptune

The for loop specifies:

1} The variable name to use.

2} The set of values to loop over.

3} The word “in” is used to link them together, which the objective is to become any object that supports iteration. Basically, if it can be thought of as a group of things, the user can probably loop over it.

* In addition to lists, the user can iterate over the elements of a tuple:

multiplicands = (2, 2, 2, 3, 3, 5)

product = 1

for mult in multiplicands:

product = product \* mult

product

360

* The user can even loop through each character in a string:

s = 'steganograpHy is the practicE of conceaLing a file, message, image, or video within another fiLe, message, image, Or video.'

msg = ''

# print all the uppercase letters in s, one at a time

for char in s:

if char.isupper():

print(char, end='')

HELLO

range() function:

range() is a function that returns a sequence of numbers. It turns out to be very useful for writing loops. For example, if we want to repeat some action five times:

for i in range(5):

print("Doing important work. i =", i)

Doing important work. i = 0

Doing important work. i = 1

Doing important work. i = 2

Doing important work. i = 3

Doing important work. i = 4

while loops:

The other type of loop in Python is a while loop, which iterates until some condition is met:

i = 0

while i < 10:

print(i, end=' ')

i += 1 # increase the value of i by 1

0 1 2 3 4 5 6 7 8 9

The argument of the while loop is evaluated as a Boolean statement, and the loop is executed until the statement evaluates to False.

List comprehensions:

* List comprehensions are one of Python's most beloved and unique features. The easiest way to understand them is probably to just look at a few examples:

squares = [n\*\*2 for n in range(10)]

squares

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

* Here's how we would do the same thing without a list comprehension:

squares = []

for n in range(10):

squares.append(n\*\*2)

squares

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

* We can also add an if condition:

short\_planets = [planet for planet in planets if len(planet) < 6]

short\_planets

['Venus', 'Earth', 'Mars']

Here's an example of filtering with an if condition and applying some transformation to the loop variable:

# str.upper() returns an all-caps version of a string

loud\_short\_planets = [planet.upper() + '!' for planet in planets if len(planet) < 6]

loud\_short\_planets

['VENUS!', 'EARTH!', 'MARS!']

Programmers usually write these on a single line, but the user might find the structure clearer when it's split up over 3 lines:

[

planet.upper() + '!'

for planet in planets

if len(planet) < 6

]

['VENUS!', 'EARTH!', 'MARS!']

* The expression on the left doesn't technically have to involve the loop variable (though it'd be pretty unusual for it not to). What do you think the expression below will evaluate to?

[32 for planet in planets]

[32, 32, 32, 32, 32, 32, 32, 32]

* List comprehensions combined with functions like min, max, and sum can lead to impressive one-line solutions for problems that would otherwise require several lines of code. For example, compare the following two cells of code that do the same thing.

def count\_negatives(nums):

"""Return the number of negative numbers in the given list.

>>> count\_negatives([5, -1, -2, 0, 3])

2

"""

n\_negative = 0

for num in nums:

if num < 0:

n\_negative = n\_negative + 1

return n\_negative

Here's a solution using a list comprehension:

def count\_negatives(nums):

return len([num for num in nums if num < 0])

Much better, right?

* Well if all we care about is minimizing the length of our code, this third solution is better still!

def count\_negatives(nums):

# Reminder: in the "booleans and conditionals" exercises, we learned about a quirk of

# Python where it calculates something like True + True + False + True to be equal to 3.

return sum([num < 0 for num in nums])

Which of these solutions is the "best" is entirely subjective. Solving a problem with less code is always nice, but it's worth keeping in mind the following lines from The Zen of Python:

1} Readability counts.

2} Explicit is better than implicit.

So, use these tools to make compact readable programs. But when you have to choose, favor code that is easy for others to understand.