The “**for**”statement in Python

Nick Szirbik

Sept. 2020

Week 2 of the DAPOM course (3)

A **for** loop is used for iterating over a collection (that is either a list, a tuple, a dictionary, a set, or a string).

- Looping through a string:

```
for each_character in "banana":
  print(each_character)
```


- Looping through a list:


```
fruits = ["apple","banana","cherry","orange","kiwi","melon","mango"]
```



```
for each_fruit in fruits:
  print(each_fruit)
```

- Looping through a range:

```
for x in range(6):
  print(x)
```

Note that range(6) is not giving the values of 0 to 6, but the values 0 to 5.

…try using range(3,8), or range (10, 30, 3)

Jumping out and over of loops (break and continue)

- Using break, we can end a loop if a condition is satisfied,

before it ends normally, e.g.:

```
fruits = ["apple","banana","pear","orange","kiwi","melon","mango"]
for fruit in fruits:
  print(fruit)
  if fruit == "banana":
    break
```


- With the continue statement we can stop the current iteration of the loop, and continue with the next:

```
for fruit in fruits:
  if fruit == "banana":
    continue
  print(fruit)
```

...this last code does not print banana, but prints all the rest

Nested loops

- Making 9 combinations of two lists of three elements:

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]
for adjective in adj:
  for fruit in fruits:
    print(adjective, fruit)
```


- Creating all numbers from 0 to 99, by concatenating two one-digit strings:

```
for first_digit in range(10):
    for second_digit in range(10):
        print(str(first_digit)+str(second_digit))
```

A few more tricks to be used with **for**:

- Doing something when the loops ends:

```
for x in range(67777):
  print(x)
else:
  print("Finally finished!")
```


- for loops cannot be empty, but if you for some reason during the development of code have a for loop with no content, put in the pass statement to avoid getting an error.

```
for x in [0, 1, 2]:
  pass
```

List comprehensions in Python

- Instead of writing a lot of code to produce a new list from an old list, like this:


```
new_things = []
for ITEM in old_things:
        if condition_based_on(ITEM):
        new_things.append("something with " + ITEM)
```


- We can write in Python much more compact code:

```
new_things = ["something with " + ITEM for ITEM in old_things if
condition_based_on(ITEM)]
```

Example (double the odd numbers in a list and make a new list)

- Old programming style (as one would do in C, C++, or Java):


```
numbers = [1, 2, 3, 4, 5]
doubled_odds = []
for n in numbers:
    if n % 2 == 1:
        doubled_odds.append(n * 2)
```


- Pythonesque style:

```
numbers = [1, 2, 3, 4, 5]
doubled_odds = [n * 2 for n in numbers if n % 2 == 1]
```