CptS 111, Spring 2023
Lect. #16, Mar. 20, 2023
Class Notes

Today's Agenda:

1. Three approaches for itemization using `for` -loops
2. Quick review of data structures in Python
3. Using dictionaries as iterables in iterating `for` -loops
4. Using the `zip()` function to create a dictionary

Ch. 6 (cont.)

Loops (cont.)

1. Numbering Using Counting and Iterating `for` -loops and `enumerate()`

Suppose we have the list `goodies = ['croissants', 'cookies', 'coffee ice cream', 'scones']` and we want to create an itemized list (not a Python data structure), i.e.,

```
1. croissants
2. cookies
3. coffee ice cream
4. scones
```

from this list. We can use a `for` -loop to accomplish this three different ways.

In [1]:
```
# Method 1: Counting for-loop
# Note that we need to use i+1 so the list starts with 1.

goodies = ['croissants', 'cookies', 'coffee ice cream', 'scones']
for i in range(len(goodies)):
    print(f'{i+1}. {goodies[i]}')
```

```
1. croissants
2. cookies
3. coffee ice cream
4. scones
```

In [2]:
```
# Method 2: Iterating for-loop
# Note that we need to add an extra variable i

i = 1
for goodie in goodies:
    print(f'{i}. {goodie}')
    i += 1
```

```
1. croissants
2. cookies
3. coffee ice cream
4. scones
```

For the iterating `for` -loop, we had to add an extra variable `i` in order to create an itemized list. To avoid this, we can use the `enumerate()` function with the iterable as its argument. The `enumerate()` function provides both the index and the element at each iteration.

Template for iterating `for` -loop with `enumerate()` :

```
for <index>, <item> in enumerate(<iterable>):
    <loop_body>
```

In [3]:
```
# Method 3: Using enumerate()
# Note that we have to use i+1 so the list starts with 1.

for i, goodie in enumerate(goodies):     # i is the list index
    print(f'{i+1}. {goodie}')
```

```
1. croissants
2. cookies
3. coffee ice cream
4. scones
```

For both the counting `for` -loop and the iterating `for` -loop with the `enumerate()` function, we need to use `i+1` . If we don't, our lists will begin with a 0.

In [4]:
```
# Using enumerate() without adding 1 to i
# Itemization will start at 0

for i, goodie in enumerate(goodies):
    print(f'{i}. {goodie}')
```

```
0. croissants
1. cookies
2. coffee ice cream
3. scones
```

2. Brief Review of Data Structures in Python

Recall that Python has five built-in data structures (ways of structuring data so that access to data and modification of data are easy):

`strings, lists, tuples, sets, dictionaries`

1. All five are containers and iterables
2. Lists, sets, and dictionaries are mutable
3. Strings and tuples are immutable
4. Lists, strings, and tuples are sequences; sets and dictionaries aren't

Data Structure	Iterable	Sequence	Mutable	Creation	Example
list	yes	yes	yes	`list()`	[1, 1.618, ['a', 'b']]
set	yes	no	yes	`set()`	{1, 1.618, 'a', 'b'}
tuple	yes	yes	no	`tuple()`	(1, 1.618, ['a', 'b'])
string	yes	yes	no	`str()`	'The Cat In The Hat'
dictionary	yes	no	yes	`dict()`	{1:'one', 'two':2, (1, 2):'tuple'}

Dictionaries are made up of key-value pairs. Keys must be immutable, but values can be anything, including other dictionaries.

We can use strings, lists, tuples, and sets as iterables in an iterating `for` -loop all in the same way

In [5]:
```
# String as iterable

for ch in 'goodies':
    print(ch, end=' * ')
```

```
g * o * o * d * i * e * s *
```

In [6]:
```
# List as iterable

for goodie in goodies:
    print(goodie)
```

```
croissants
cookies
coffee ice cream
scones
```

In [7]:
```
# Tuple as iterable

goodies_tuple = ('croissants', 'cookies', 'coffee ice cream', 'scones')
for goodie in goodies_tuple:
    print(goodie)
```

```
croissants
cookies
coffee ice cream
scones
```

In [10]:
```
# Set as iterable
# Note that because sets aren't sequences, they aren't necessarily
# accessed sequentially

goodies_set = {'croissants', 'cookies', 'coffee ice cream', 'scones'}
for goodie in goodies_set:
    print(goodie)
```

```
croissants
cookies
coffee ice cream
scones
```

We cannot, however, use dictionaries as iterables in the same way.

In [11]:
```
# Dictionary as iterable
# End up with keys, not values

goodies_dict = {'flaky': 'croissants', 'chewy': 'cookies',
                'creamy': 'coffee ice cream', 'rich': 'scones'}
for goodie in goodies_dict:
    print(goodie)
```

```
flaky
chewy
creamy
rich
```

This wasn't exactly what we wanted! Instead we have to do the following.

In [12]:
```
# Dictionary as iterable -- correct usage
# goodie is key; goodies_dict[goodie] is value for given key

for goodie in goodies_dict:       # goodie is the key
    print()
    print('goodies_dict key:  ', goodie)
    print('goodies_dict value:', goodies_dict[goodie])   # prints value o
```

```
goodies_dict key:   flaky
goodies_dict value: croissants

goodies_dict key:   chewy
goodies_dict value: cookies

goodies_dict key:   creamy
goodies_dict value: coffee ice cream

goodies_dict key:   rich
goodies_dict value: scones
```

where `goodie` is actually the key and `goodies_dict[goodie]` is its value.

3. Using Dictionaries as Iterables in iterating `for` -loops

As we just saw, we can use the key as the loop variable, but we can also use what are termed dictionary view objects as iterables.

```
for key in dict1:                 # Keys used to get values as ab
ove
for key, value in dict1.items():  # Both keys and values
for key in dict1.keys():          # Just keys
for value in dict1.values():      # Just values
```

where the bottom three are view objects. Let's see how this works.

In [13]:
```
# Both keys and values

for key, value in goodies_dict.items():
    print(f'{key}: {value}')
```

```
flaky: croissants
chewy: cookies
creamy: coffee ice cream
rich: scones
```

In [14]:
```
# Just values

for value in goodies_dict.values():
    print(value)
```

```
croissants
cookies
coffee ice cream
scones
```

In [15]:
```
# Just keys

for key in goodies_dict.keys():
    print(key)
```

```
flaky
chewy
creamy
rich
```

4. Using `zip()` to Create a Dictionary

We can create a dictionary from two lists using the `zip()` function as follows. Don't confuse this function with compressing external files to create a .zip file!

In [16]:
```
# Using zip() to create a dictionary from two lists

colors = ['red', 'orange', 'pink', 'white', 'yellow']           # list 1
flowers = ['roses', 'lilies', 'tulips', 'alyssum', 'daffodils'] # list 2
catalog = dict(zip(colors, flowers)) # zip lists together then create dic
catalog
```

Out[16]:
```
{'red': 'roses',
 'orange': 'lilies',
 'pink': 'tulips',
 'white': 'alyssum',
 'yellow': 'daffodils'}
```

In [17]:
```
# Let's try something a little fancier
# Use list(range()) to create a list of integers; make it start at 1
# Assume we don't know how long the flowers list is, so use its length
# Add 1 to len(flowers) because we started at 1, not 0

flowers = ['roses', 'lilies', 'tulips', 'alyssum', 'daffodils', 'dahlias'
item_no = list(range(1, len(flowers)+1))
print(item_no)
catalog = dict(zip(item_no, flowers)) # zip lists together then create di
catalog
```

```
[1, 2, 3, 4, 5, 6]
```

Out[17]:
```
{1: 'roses',
 2: 'lilies',
 3: 'tulips',
 4: 'alyssum',
 5: 'daffodils',
 6: 'dahlias'}
```

Pretty cool, eh?