for loops – iterating over items (class slides)

CSc 110 - the other type of `for` loop

Review of `for in range():`

```
for n in range(5):
  print(n)
```

```
0
1
2
3
4
```

```
numbers = [2, 1, 4, 6, 23, 2]
for i in range(len(numbers)):
  print(numbers[i])
```

```
2
1
4
6
23
2
```

Introducing `for x in list:`

```
numbers = [2, 1, 4, 6, 23, 2]
for n in numbers:
  print(n)
```

```
2
1
4
6
23
2
```

Write a function

1. Its name is `count_vowels`
2. It takes a `string` argument
3. It creates a `dictionary`
4. It returns the `dictionary` with the count of every lowercase vowel in `string`

```
assert count_vowels("") == {"a": 0, "e": 0, "i": 0, "o": 0, "u": 0}
assert count_vowels("banana") == {"a": 3, "e": 0, "i": 0, "o": 0, "u": 0}
```

Write a function – solution

```
def count_vowels(string):
  counts = {"a": 0, "e": 0, "i": 0, "o": 0, "u": 0}
  for char in string:
    if char in counts:
      counts[char] += 1
  return counts

def main():
  assert count_vowels("") == {"a": 0, "e": 0, "i": 0, "o": 0, "u": 0}
  assert count_vowels("banana") == {"a": 3, "e": 0, "i": 0, "o": 0, "u": 0}
  print("Passed all tests.")

main()
```

```
Passed all tests.
```

Write a function

1. Its name is `count_chars`
2. It takes a `string` argument
3. It creates a `dictionary`
4. It returns the `dictionary` with the count of every characters in `string`

```
assert count_chars("") == {}
assert count_chars("banana") == {"b": 1, "a": 3, "n": 2}
```

Write a function – solution

```
def count_chars(string):
  counts = {}
  for char in string:
    if char in counts:
      counts[char] += 1
    else:
      counts[char] = 1

  return counts

def main():
  assert count_chars("") == {}
  assert count_chars("banana") == {"b": 1, "a": 3, "n": 2}
  print("Passed all tests.")

main()
```

```
Passed all tests.
```

Write a function

1. Its name is `tally_negatives`
2. It takes a list of `numbers` as argument
3. It returns a dictionary that maps each negative value in `numbers` to its frequency in `numbers`

Test cases:

```
assert  tally_negatives([1, -2, 0, -4, -2]) == {-2: 2, -4: 1}
assert tally_negatives([]) == {}
```

Write a function – solution

```
def tally_negatives(numbers):
  tally = {}
  for n in numbers:
    if n < 0:
      if n not in tally:
        tally[n] = 0
      tally[n] += 1
  return tally

def main():
  assert  tally_negatives([1, -2, 0, -4, -2]) == {-2: 2, -4: 1}
  assert tally_negatives([]) == {}
  print("Passed all tests.")

main()
```

```
Passed all tests.
```

What happens when we do `for k in dictionary:`

```
scores = {'A': 10, 'B': 25, 'C': 27, 'D': 10, 'E': 5}
for key in scores:
  print(key)
```

```
A
B
C
D
E
```

Methods for dictionaries

We can use `.values()` to get only the values in a dictionary:

```
scores = {'A': 10, 'B': 25, 'C': 27, 'D': 10, 'E': 5}
scores.values()
```

```
dict_values([10, 25, 27, 10, 5])
```

We can use `.keys()` to get only the keys in a dictionary:

```
scores = {'A': 10, 'B': 25, 'C': 27, 'D': 10, 'E': 5}
scores.keys()
```

```
dict_keys(['A', 'B', 'C', 'D', 'E'])
```

We can use `.items()` to get tuples for keys and values:

```
scores = {'A': 10, 'B': 25, 'C': 27, 'D': 10, 'E': 5}
scores.items()
```

```
dict_items([('A', 10), ('B', 25), ('C', 27), ('D', 10), ('E', 5)])
```

`for x in list`

```
scores = {'A': 10, 'B': 25, 'C': 27, 'D': 10, 'E': 5}
for value in scores.values():
  print(value)
```

```
10
25
27
10
5
```

```
scores = {'A': 10, 'B': 25, 'C': 27, 'D': 10, 'E': 5}
for key in scores.keys():
```

```
  print(key)
```

```
A
B
C
D
E
```

`for key, value in dictionary.items()`

```
scores = {'A': 10, 'B': 25, 'C': 27, 'D': 10, 'E': 5}
for key, value in scores.items():
  print(key, value)
```

```
A 10
B 25
C 27
D 10
E 5
```

Write a function

1. Its name is `keys_and_values`
2. It takes a `dictionary` as argument
3. It returns a list with all the keys and values in the `dictionary`

Test cases:

```
assert keys_and_values({'A': 10, 'B': 25, 'C': 27, 'D': 10, 'E': 5}) == \
    ['A', 10, 'B', 25, 'C', 27, 'D', 10, 'E', 5]
```

Write a function – solution

```
def keys_and_values(dictionary):
  new_list = []
  for key, value in dictionary.items():
    new_list.append(key)
    new_list.append(value)
```

```
  return new_list

def main():
  assert keys_and_values({'A': 10, 'B': 25, 'C': 27, 'D': 10, 'E': 5}) == ['A', 10, 'B', 2

main()
```

Write a function

1. Its name is `merge_dictionaries`
2. It takes two arguments: `dict_1` and `dict_2`
3. It mutates `dict_1`, by adding to it all key-values pairs in `dict_2`
4. If a key is in both dictionaries, the values are added
5. Don't use `.update()`

Test cases:

```
dict_1 = {"a": 20, "e": 5}
dict_2 = {"e": 10, "i": 2}
assert merge_dictionaries(dict_1, dict_2) == {"a": 20, "e": 15, "i": 2}
```

Write a function – solution

```
def merge_dictionaries(dict_1, dict_2):
  for key, value in dict_2.items():
    if key in dict_1:
      dict_1[key] += value
    else:
      dict_1[key] = value
  return dict_1

def main():
  dict_1 = {"a": 20, "e": 5}
  dict_2 = {"e": 10, "i": 2}
  assert merge_dictionaries(dict_1, dict_2) == {"a": 20, "e": 15, "i": 2}

main()
```