Computational Thinking with Programming

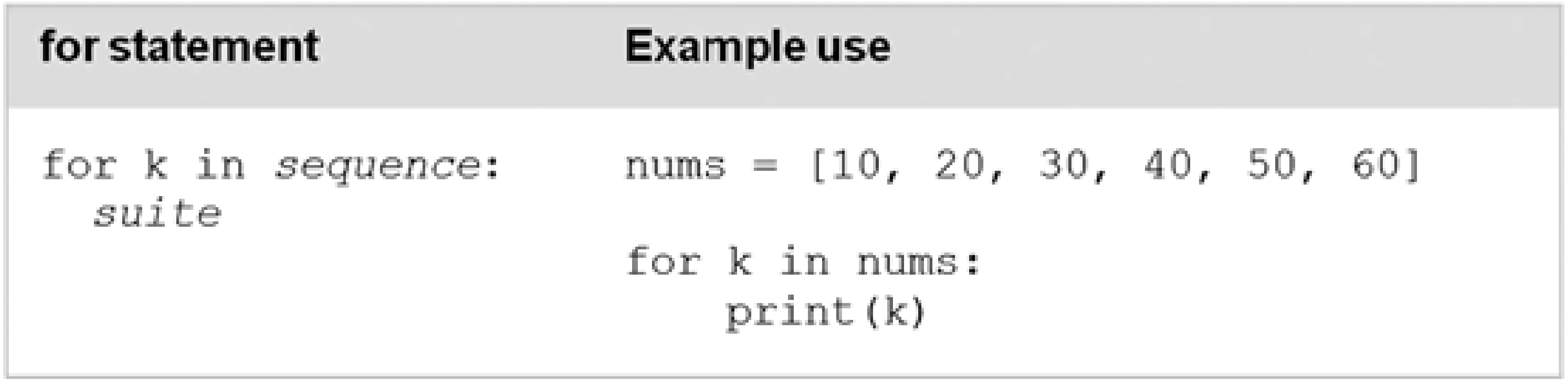
@cse_bennett @csebennett

Iterating Over Lists

- Python’s for statement provides a convenient means of iterating over lists (and other sequences).
- There are three ways we can do it.
    - For loops
    - While loops

Iterating Over Lists using for loops

- A **for statement** is an iterative control statement that iterates once for each element in a specified sequence of elements. Thus, for loops are used to construct definite loops.



for statement	Example use

`for k in` *`sequence`*`:`
    `suite`
	`nums = [10, 20, 30, 40, 50, 60]`
	
	`for k in nums:`
	    `print(k)`

- Variable k is referred to as a **loop variable**. Since there are six elements in the provided list, the for loop iterates exactly six times.

Iterating Over Lists using while loop

- To contrast the use of for loops and while loops for list iteration, the same iteration is provided as a while loop below,

```
k = 0
while k < len(nums):
    print(nums[k])
    k = k + 1
```

- In the while loop version, loop variable k must be initialized to 0 and incremented by 1 each time through the loop.
- In the for loop version, loop variable k automatically iterates over the provided sequence of values.

Use of for loop in iteration

- The for statement can be applied to all sequence types, including strings. Thus, iteration over a string can be done as follows (which prints each letter on a separate line).

```
for ch in 'Hello':
    print(ch)
```

Built-in range Function

- Python provides a built-in range function that can be used for generating a sequence of integers that a for loop can iterate over, as shown below.

```
sum = 0
for k in range(1, 11):
    sum = sum + k
```

Iterating Over List Elements vs. List Index Values

- When the elements of a list need to be accessed, but not altered, a loop variable that iterates over each list element is an appropriate approach. However, there are times when the loop variable must iterate over the *index values* of a list instead.

Loop variable iterating over the elements of a sequence

```
nums = [10, 20, 30, 40, 50, 60]

for k in nums:
    sum = sum + k
```

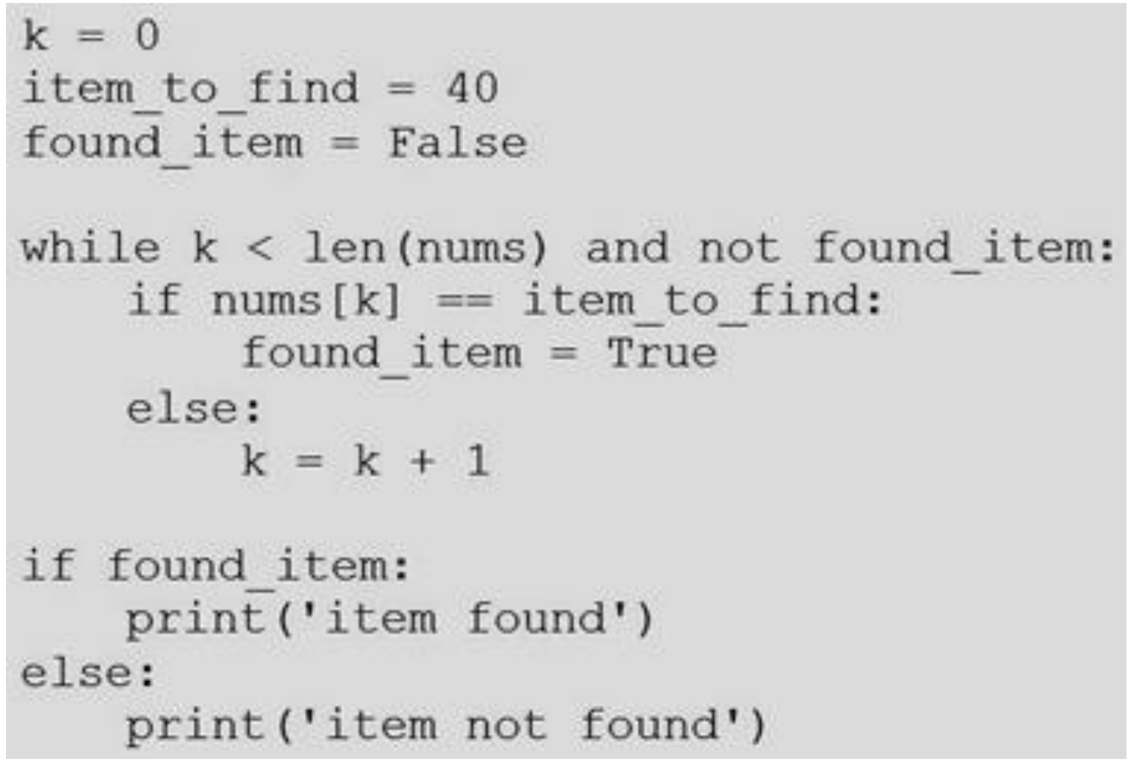
Loop variable iterating over the index values of a sequence

```
nums = [10, 20, 30, 40, 50, 60]

for k in range(len(nums)):
    sum = sum + nums[k]
```

While Loops and Lists

- There are situations in which a sequence is to be traversed while a given condition is true. In such cases, a while loop is the appropriate control structure.



```
k = 0
item_to_find = 40
found_item = False

while k < len(nums) and not found_item:
    if nums[k] == item_to_find:
        found_item = True
    else:
        k = k + 1

if found_item:
    print('item found')
else:
    print('item not found')
```

Lists – A summary

Create a new list

- # empty list

my_list = []

- # list of integers

my_list = [1, 2, 3]

- # list with mixed datatypes

my_list = [1, "Hello", 3.4]

# nested list

my_list = [[1,2,3], [8, 4, 6], [6,6,7]]

my_list = [[1,2,3], [8, 4, 6], [6,6,7]]

Access A List/ List Index

```
my_list = ['p','r','o','b','e’]
print(my_list[0])
# Output: p

print(my_list[2])
# Output: o

print(my_list[4])
Output: e
```

```
print(my_list[4.5])
# Error! Only integer can be used
for indexing

# Nested List
n_list = ["Happy", [2,0,1,5]]
print(n_list[0][1])
# Output: a

print(n_list[1][3])
# Output: 5
```

Negative Indexing

```
my_list = ['p','r','o','b','e’]
print(my_list[-1])
# Output: e

print(my_list[-5])
# Output: p
```

Slicing

```
my_list = [3, 4, 2, 5, 6, 8, 9, 1, 5, 7, 0, 11, 45, 32, 90]
# print elements 3rd to 5th
print(my_list[2:5])
# elements beginning to 4th
print(my_list[:4])
# elements 6th to end
print(my_list[5:])
# elements beginning to end
print(my_list[:])
```

Change elements to a list

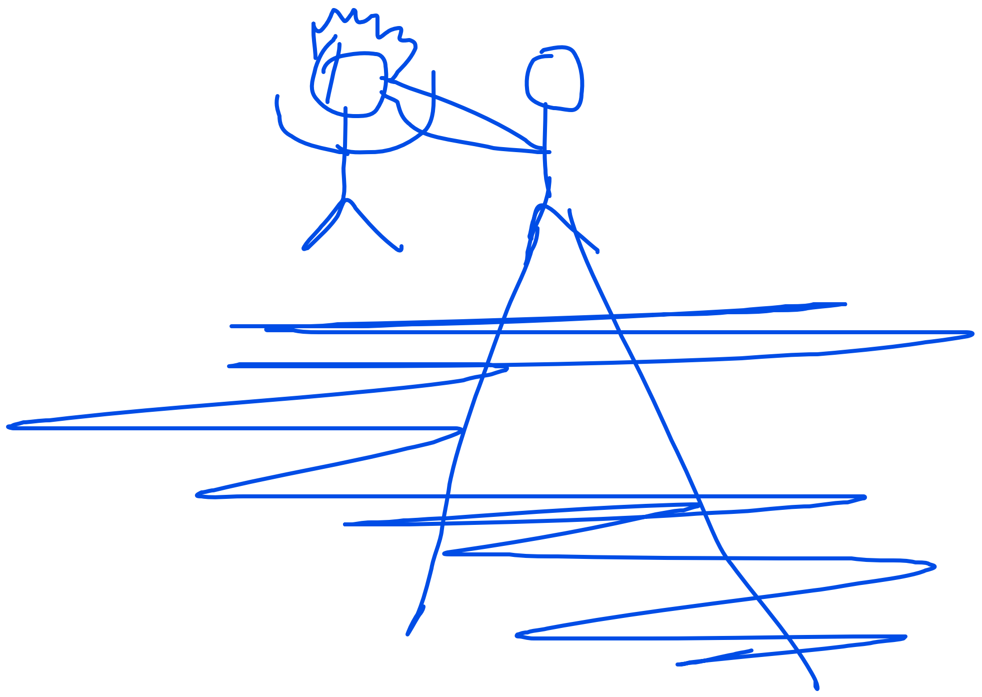
```
my_list = [2, 4, 6, 8]
# change the 1st item
my_list[0] = 1
print(my_list )
# Output: [1, 4, 6, 8]
# change 2nd to 4th items
my_list[1:4] = [3, 5, 7]
print(my_list )
# Output: [1, 3, 5, 7]
```

Add element in the list

```
odd = [1, 3, 5]
odd.append(7)
print(odd)
# Output: [1, 3, 5, 7]
odd.extend([9, 11, 13])
print(odd)
# Output: [1, 3, 5, 7, 9, 11, 13]
```

Insert an Element

```
odd = [1, 9]
odd.insert(1,3)
print(odd)
# Output: [1, 3, 9]
odd[2:2] = [5, 7]
print(odd)
# Output: [1, 3, 5, 7, 9]
```



Delete elements

```
my_list = ['p','r','o','b','l','e','m’]
# delete one item
del my_list[2]
print(my_list)
# Output: ['p', 'r', 'b', 'l', 'e', 'm']

# delete multiple items
del my_list[1:5]
print(my_list)
# Output: ['p', ‘m’]

# delete entire list
del my_list
```

Remove an element, pop, clear

```
my_list = ['p','r','o','b','l','e','m’]
my_list.remove(‘p’)
print(my_list)
# Output: ['r', 'o', 'b', 'l', 'e', 'm’]

print(my_list.pop(1))
# Output: ‘o’

print(my_list)
# Output: ['r', 'b', 'l', 'e', 'm’]
```

```
print(my_list.pop())
# Output: ‘m’

print(my_list)
# Output: ['r', 'b', 'l', 'e’]

my_list.clear()
print(my_list)
# Output: []
```

Index, count

```
my_list = [3, 8, 1, 6, 0, 8, 4]
print(my_list.index(8))
# Output: 1

print(my_list.count(8))
# Output: 2
```

- colours = ["red", "green", "blue", "green", "yellow”, ”white”]
- print(colours.index("green"))
- print(colours.index("green", 2))
- print(colours.index("green", 2,5))
- print(colours.index("black"))

Sort, Reverse

```
my_list = [3, 8, 1, 6, 0, 8, 4]
my_list.sort()
print(my_list)
# Output: [0, 1, 3, 4, 6, 8, 8]

my_list = [3, 8, 1, 6, 0, 8, 4]
my_list .sort(reverse=True)
print(my_list)
# Output: [ 8, 8,6,4,3,1,0]
```

Membership function

```
my_list = ['p','r','o','b','l','e','m’]
print('p' in my_list)
# Output: True

print('a' in my_list)
# Output: False

print('c' not in my_list)
# Output: True
```