# Lists

## COMP0015 Introduction to Programming

Rae Harbird, r.harbird@ucl.ac.uk

Lists

Tuples

Questions

# Lists

# What is a list?

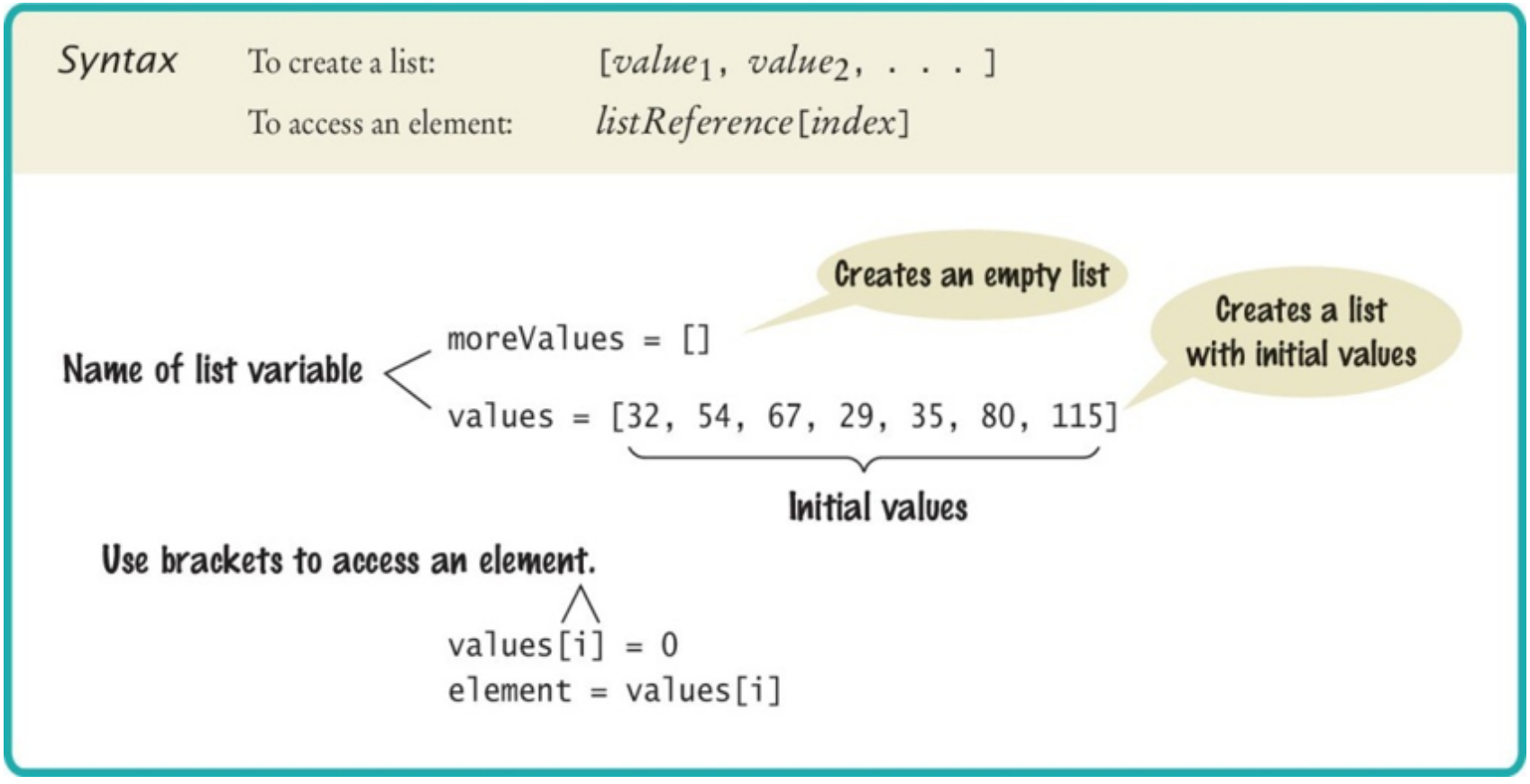
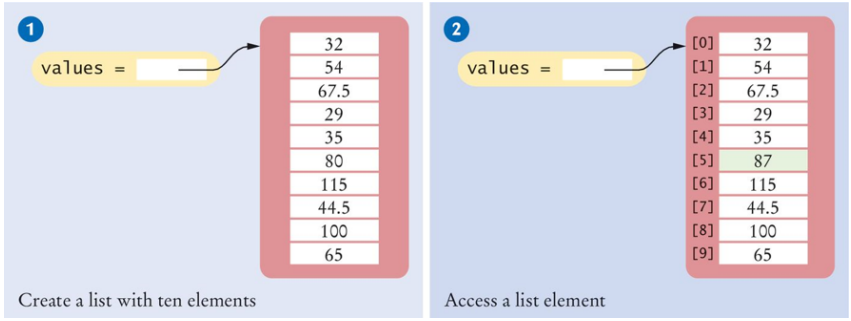



*Image: Chapter 6, Python for Everybody, 2nd Ed.*

# Create and Use a List




```
>>> values = [32, 54, 67.5. 29, 35, 80, 115, 44.5, 100, 65]
>>> print(values[5])
80
>>>
```

Set the value of the 6th element to 87

```
>>> values[5] = 87
>>> print(values)
[32, 54, 67.5, 29, 35, 87, 115, 44.5, 100, 65]
>>>
```

# Another example

```
# A doctor has 10 patients
patientAgeList = [2, 5, 92, 27, 73, 14, 58, 44, 67, 10]

# Remember, count elements from 0
print("patient's age in 0th position: ", patient[0])
print("patient's age in 2nd position: ", patient[2])

# Change the value of any list element, assign a new value
patient[2] = 82
```

# Slices and Lists

- Just like strings, you can specify a slice of a list.
- The result is a new list containing the slice you wanted.

```
# A doctor has 10 patients
patientAgeList = [2, 5, 92, 27, 73, 14, 58, 44, 67, 10]

# Remember, count elements from 0

# Print the first 4 elements in the list
print("First 4 patients: ", patient[0:4])

# print the last 4 elements in the list
print("Last 4 patients: ", patient[-4:])
```

# Traverse a List

There is more than one way to iterate over a list with a `for` loop. You can traverse the list using the element numbers but Python has a useful shorthand.

```
# A doctor has 10 patients
patientAgeList = [2, 5, 92, 27, 73, 14, 58, 44, 67, 10]

totalAge = 0

# Use range() with len() to find the elements.
for counter in range(len(patientAgeList)) :
    totalAge += patientAgeList[counter]

# Use Python's "in" operator
for age in patientAgeList :
    totalAge += age

print("Average patient's age: {:.2f}".format(totalAge
                                             / len(patientAgeList))
```

# Which `for` loop?

**Caution!!!**

Don't use this form of loop when you want to change the value of elements in a list:

```
# Code to add 1 to every age in the list.
# .. BEWARE!!!!! This doesn't work.
patientAgeList = [2, 5, 92, 27, 73, 14, 58, 44, 67, 10]

for age in patientAgeList :
    age = age + 1

print("Patient ages are: ", patientAgeList)
```

See: **Section 10.3, Think Like a Computer Scientist**

# Which `for` loop? (cont.)

You must use this form when changing the elements:

```
# use this form and assign the new value to the element at the index.
patientAgeList = [2, 5, 92, 27, 73, 14, 58, 44, 67, 10]

for i in range(len(patientAgeList)):
    patientAgeList[i] = patientAgeList[i] + 1
```

# When you should not use a for loop at all

Don't use a `for` loop when you are removing elements from the list as you are iterating over it. Imagine that you want to remove all words from a list of words that have less than 4 letters.

```
# Code to remove words less than 4 characters long.
# .. BEWARE!!!!! This doesn't work.

words = ["Welcome", "to", "the", "island!"]

print("\n", words)

for i in range(len(words)):
    word = words[i]
    print("\ti: {}, {}".format(i, words[i]))
    if len(word) < 4 :
        words.pop(i)

print("\n", words)
```

# What's the problem?

- There is a subtle problem with the previous example. At the start of the `for` loop, Python uses the list length to determine how many times to perform the loop. If you change the size of the list then Python will try to process elements beyond the new list length.
- Try it out for yourself. There is a program called `delete_small_words_with_for.py` in the `src` folder.

# The solution

Here is `delete_small_words_with_while.py`.

```
words = ["Welcome", "to", "the", "island!"]

print("\n", words)
i = 0
print()
while i < len(words) :
    print("\ti: {}, {}".format(i, words[i]))
    word = words[i]
    if len(word) < 4 :
        words.pop(i)
    else :
        i = i + 1

print("\n", words)
```

# More List Operations

- Find, use the `in` operator:

```
if "Cindy" in friends :
   print("She's a friend")
```

- Append

```
friends = []

friends.append("Harry")
friends.append("Emily")
friends.append("Bob")
friends.append("Cari")
```

- Insert

```
friends = ["Harry", "Emily", "Bob", "Cari"]
friends.insert(1, "Cindy") # add an element in the 1st position
friends.insert(5, "Bill")  # Same as append
```

# More Operations

- Concatenation

```
myFriends = ["Fritz", "Cindy"]
yourFriends = ["Lee", "Pat", "Phuong"]
ourFriends = myFriends + yourFriends
# Sets ourFriends to ["Fritz", "Cindy", "Lee", "Pat", "Phuong"]
```

- Replication One common use of replication is to initialize a list with a fixed value. For example,

```
monthlyScores = [0] * 12
# The list is [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

- Equality You can use the == operator to compare whether two lists have the same elements, in the same order. For example:

```
[1, 4, 9] == [1, 4, 9]
```

is `True`

```
[1, 4, 9] == [4, 1, 9]
```

is `False`

# Python tutorial lists

The Python tutorial contains two sections on lists

# Tuples

# What is a tuple?

A tuple is special type of list that you can't change. It's immutable. Tuples are often used to provide multiple return values from a function.

You create and access a tuple in much the same way as you do with a list. Tuples have round brackets though:

```
myTuple = (5, 10, 15)

# If you prefer, you can omit the parentheses:

anotherTuple = 5, 10, 15
```

# More Tuples

```
# Use square brackets for accessing elements
element = myTuple[1]

# Find the number of elements with the len function.
lengthOfTuple = len(myTuple)

# Iterate over the elements of a tuple using for loops.
for thing in myTuple:
    print(thing)

# Test for members using the in and not in operators
if myNumber not in myTuple:
    print("Not found.")
```

# List Operations I

**Table 1 Common List Functions and Operators**

| Operation | Description |
|---|---|
| `[]`<br>`[`$elem_1$, $elem_2$, ..., $elem_n$`]` | Creates a new empty list or a list that contains the initial elements provided. |
| `len(`*l*`)` | Returns the number of elements in list *l*. |
| `list(`*sequence*`)` | Creates a new list containing all elements of the sequence. |
| `values * num` | Creates a new list by replicating the elements in the `values` list `num` times. |
| `values + moreValues` | Creates a new list by concatenating elements in both lists. |

*Image: Chapter 6, Python for Everyone, 2nd Ed.*

# List Operations II

Table 1 Common List Functions and Operators

| Operation | Description |
|---|---|
| `l[from : to]` | Creates a sublist from a subsequence of elements in list *l* starting at position `from` and going through but not including the element at position `to`. Both `from` and `to` are optional. (See Special Topic 6.2.) |
| `sum(l)` | Computes the sum of the values in list *l*. |
| `min(l)`<br>`max(l)` | Returns the minimum or maximum value in list *l*. |
| $l_1$ `==` $l_2$ | Tests whether two lists have the same elements, in the same order. |

*Image: Chapter 6, Python for Everyone, 2nd Ed.*

# List Operations III

| Table 2 Common List Methods | |
|---|---|
| **Method** | **Description** |
| *l*.pop()<br>*l*.pop(*position*) | Removes the last element from the list or from the given position. All elements following the given position are moved up one place. |
| *l*.insert(*position*, *element*) | Inserts the element at the given position in the list. All elements at and following the given position are moved down. |
| *l*.append(*element*) | Appends the element to the end of the list. |
| *l*.index(*element*) | Returns the position of the given element in the list. The element must be in the list. |
| *l*.remove(*element*) | Removes the given element from the list and moves all elements following it up one position. |
| *l*.sort() | Sorts the elements in the list from smallest to largest. |

*Image: Chapter 6, Python for Everyone, 2nd Ed.*

# Questions

# Question 1

What is wrong with each of the following code segments?

a.

```
values = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
for i in range(1, 11) :
    values[i] = i * i
```

b.

```
values = []
for i in range(len(values)) :
    values[i] = i * i
```

*(Horstmann 370) Horstmann, Cay S., Rance Necaise. Python for Everyone, Interactive Edition, 2nd Edition. Wiley, 2016-05-09. VitalBook file.*

# Answer 1

## a.

```
values = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
for i in range(1, 11) :
    values[i] = i * i
```

The values assigned to `i` are 1 .. 10. However, the valid indices for the values list are 0 .. 9. As a result, the program will crash with an index out of bounds exception.

## b.

```
values = []
for i in range(len(values)) :
    values[i] = i * i
```

While this program will not crash, it also won't add any values to the list because `len(values)` is 0. As a result, one probably wants to change the parameter provided to range so that it is 10 instead of `len(values)`. In addition, one will need to call append inside the loop.

# Question 2

Consider the following loop for collecting all elements that match a condition; in this case, that the element is larger than 100.

```
matches = []
for element in values :
    if element > 100 :
        matches.append(element)
```

Trace the flow of the loop, where values contains the elements 110 90 100 120 80. Show two columns, for element and matches.

| Element | after loop, matches will contain |
|---|---|
| 110 | |
| 90 | |

# Answer 2

```
matches = []
for element in values :
    if element > 100 :
        matches.append(element)
```

Trace for values = [110, 90, 100, 120, 80]:

| Element | matches contains |
|---|---|
| 110 | [110] |
| 90 | [110] |
| 100 | [110] |
| 120 | [110, 120] |
| 180 | [110, 120] |