CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

http://www.cit.ie

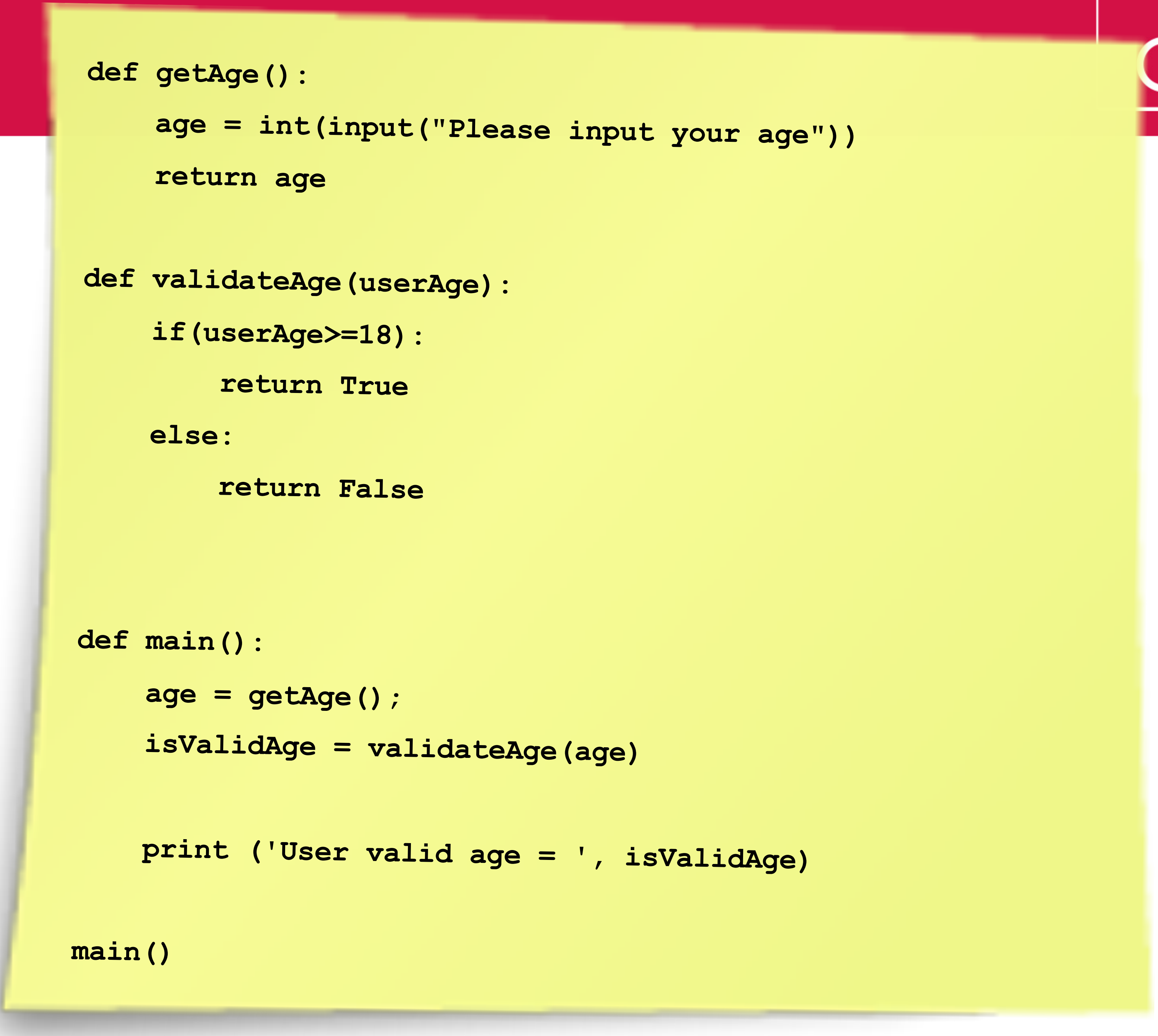
Programming for Data Analytics

Data Structures

Dr. Mohammed Hasanuzzaman
Room J102, Melbourne Building (Office)
e-mail:mohammed.hasanuzzaman@cit.ie
Website: https://mohammedhasanuzzaman.github.io

Programming Task A

- Write a method that will ask a user for their age and return the value.
- Write another method that takes in a single int parameter and returns true if the int parameter is greater than 18 or false if it is less than 18.
- Using the above methods write a program that will ask a user for their age and will inform the user if they are over 18 or not



```
def getAge():
    age = int(input("Please input your age"))
    return age

def validateAge(userAge):
    if(userAge>=18):
        return True
    else:
        return False


def main():
    age = getAge();
    isValidAge = validateAge(age)

    print ('User valid age = ', isValidAge)

main()
```

Programming Task B

- Part A
  - Write a program that will generate two random numbers between 1 and 100.
  - Write a function that will return the sum of the two numbers.
  - Write a function that will ask the user to guess the value and will return the guessed value.
  - The program should then print out a message indicating if the user has guessed correctly or not.
- Part B
  - Make this program iterative. Allow the user to continue until they chose to exit.

```
import random

def main():
    numberA = random.randint(1, 100)
    numberB = random.randint(1, 100)
    result = sum(numberA, numberB)
    guess = askUser(numberA, numberB)
    if (result == guess):
        print ("Correct")
    else:
        print ("Incorrect")


def sum(num1, num2):
    return num1+num2

def askUser(num1, num2):
    guess = int(input("What is the sum of the following numbers: "+str(num1)+" + "+str(num2)))
    return guess

main()
```

```
import random

def main():
    continueGame = 'y'
    while continueGame == 'y':
        numberA = random.randint(1, 100)
        numberB = random.randint(1, 100)
        result = sum(numberA, numberB)
        guess = askUser(numberA, numberB)
        if (result == guess):
            print ("Correct")
        else:
            print ("Incorrect")
        continueGame = input("Would you like to continue y/n")

def sum(num1, num2):
    return num1+num2

def askUser(num1, num2):
    guess = int(input("What is the sum of the following numbers: "+str(num1)+" + "+str(num2)))
    return guess

main()
```

Content

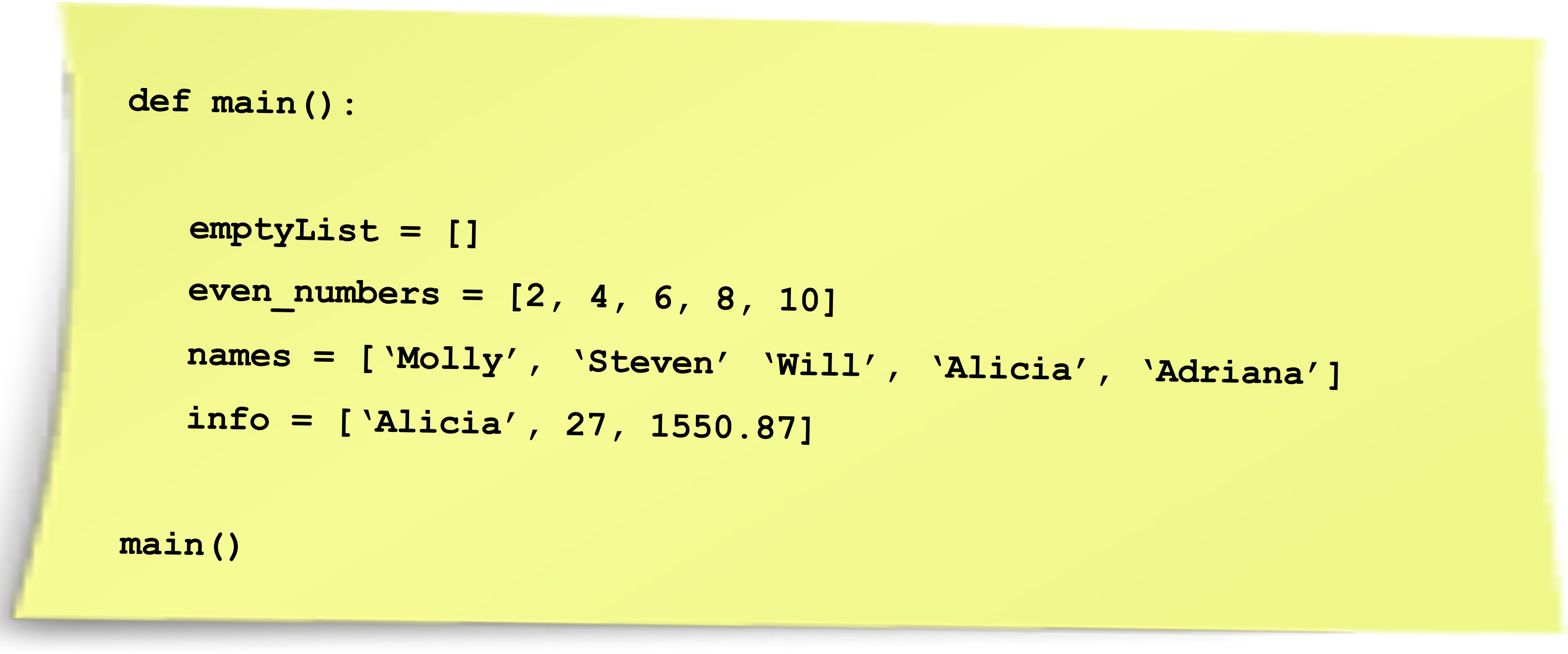
➔ <u>Lists</u>

- Dictionaries

- Sets

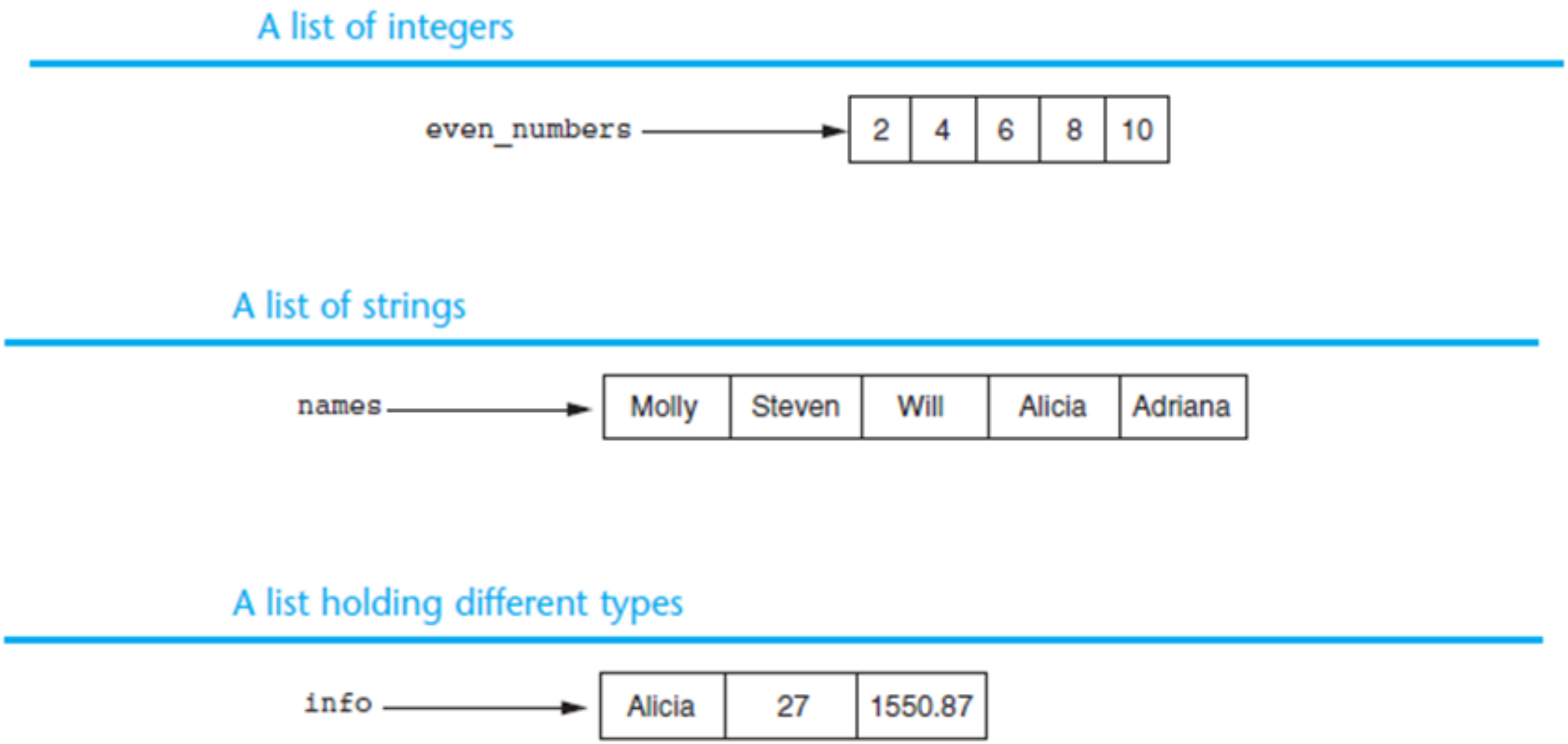
Introduction to Lists

- Lists are dynamic data structures, meaning that items may be added to them or removed from them.
- List: an object that contains multiple data items
  - Element: An item in a list
  - Format: ***`list = [item1, item2, etc]`***
  - Can hold items of different types



```
def main():

    emptyList = []
    even_numbers = [2, 4, 6, 8, 10]
    names = [‘Molly’, ‘Steven’ ‘Will’, ‘Alicia’, ‘Adriana’]
    info = [‘Alicia’, 27, 1550.87]

main()
```

Introduction to Lists (Reference Variables and Mutable Data Structure)



A list of integers

even_numbers → 2 | 4 | 6 | 8 | 10

A list of strings

names → Molly | Steven | Will | Alicia | Adriana

A list holding different types

info → Alicia | 27 | 1550.87

Iterating over a list with a for loop

```
numbers = [99, 100, 101, 102]
for num in numbers:
  print (num)
```

- If we run this code, it will print:

```
99
100
101
102
```

Indexing – Accessing an element at a position within the list

- Index: a number specifying the **position** of an element in a list
  - Enables access to an individual element in a list
  - Index of first element in the list is 0, second element is 1, and n’th element is n-1
  - numbers = [100, 200, 300, 400, 500, 600]

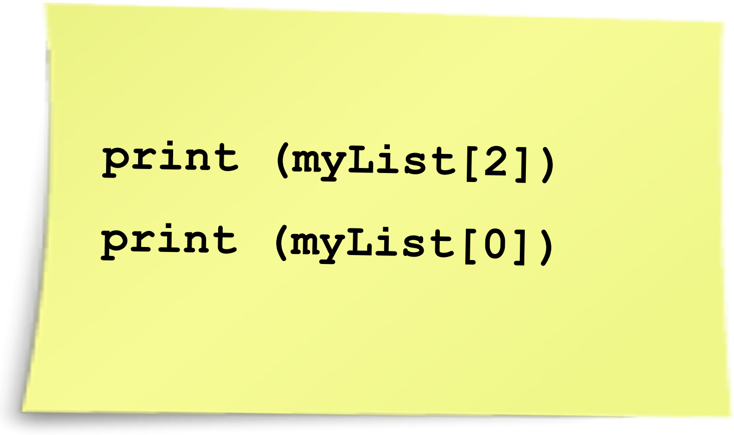
100	200	300	400	500	600
index 0	**index 1**	**index 2**	**index 3**	**index 4**	**index 5**

Indexing

myList = [l0, 20, 30, 40]

We can access the individual elements of the list with the following statement: **myList[index]**

Very important to remember that the index of the first data item starts at 0



```
print (myList[2])
print (myList[0])
```

```
30
10
```

Indexing

- Negative indexes identify positions relative to the end of the list
  - The index -1 identifies the last element, -2 identifies the next to last element, etc.

0	**0**	**0**	**0**	**0**	**0**
index -6	index -5	index -4	index -3	index -2	index -1

Indexing

myList = [l0, 20, 30, 40]

```
print (myList[-3])
print (myList[-1])
```

```
20
40
```

The len function

- len function: returns the length of a sequence such as a list
  - Example: **size = len(my_list)**

- Knowing that the len function will return length of list then how would we use it to access the last element in the list

–Returns the number of elements in the list, so the index of last element is `len(list)-1`

The len function

```
myList = [10, 20, 30]
index = 0
while index < len(myList):
    print (myList[index])
    index += 1
```

Notice we can use the len function to iterate through a list. It controls the index we access

Assigning a new value to a list

- Mutable sequence:
  - Lists are mutable, and so their **elements can be changed**

- An expression such as

```
list[i] = new_value
```

can be used to assign a new value to a list element where *i* is a valid index within the list

– Must use a valid index to prevent raising of an `IndexError` exception

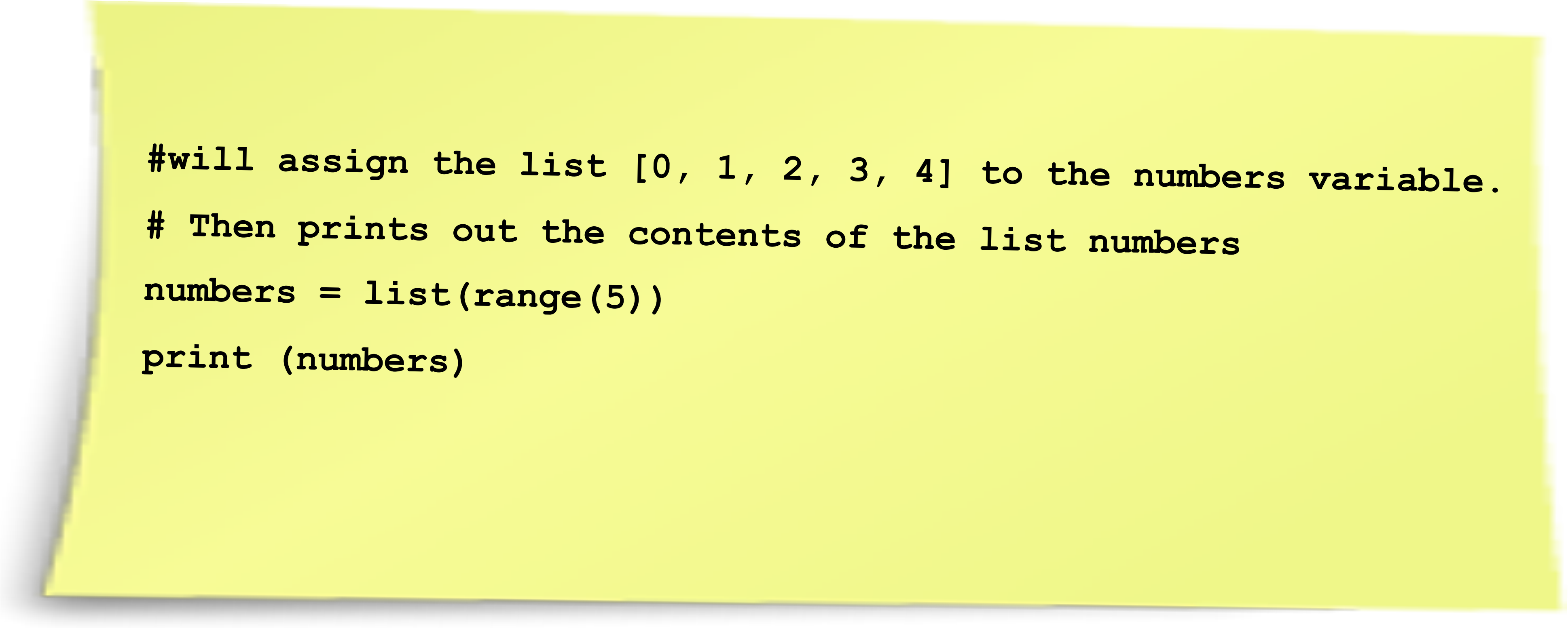
Changes Values of a List

```
numbers = [23, 54, 56, 67]
numbers[1] = 43
numbers[3] = 2
numbers[0] = 0
print (numbers)
```

The following will print out
[0, 43, 56, 2]

The Range Function

- As you will remember from the lecture notes covering iteration, the range function returns a range object that can be used to generate a list of numbers.
- We can force the range function to generate the list by passing it to a list function.



```
#will assign the list [0, 1, 2, 3, 4] to the numbers variable.
# Then prints out the contents of the list numbers
numbers = list(range(5))
print (numbers)
```

Task

- Write a program that will ask the user to enter an **upper limit for a list**

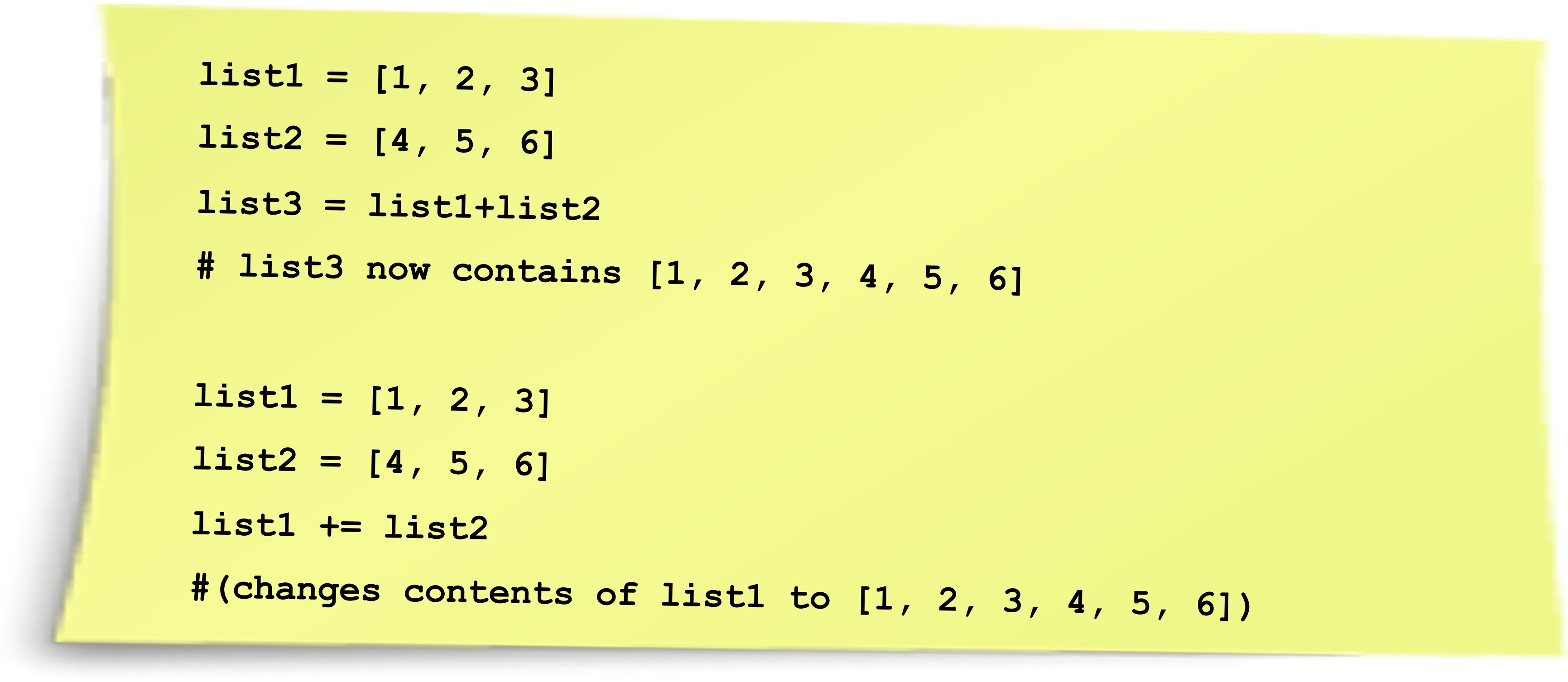
- It should then generate a list between **0 and the upper limit** and print out the contents of the list.

- Your program should then print the contents of the list in **reverse** order.

```
limit = int(input("Please enter upper limit of list"))
numbers = list(range(limit+1))
print (numbers)

index = len(numbers)-1


while index>=0:
    print (numbers[index]),
    index -= 1
```

Concatenating Lists

- <u>Concatenate</u>: join two lists together
  - The **+** operator can be used to concatenate two lists (Cannot concatenate a list with another data type, such as a number)
  - The **+=** augmented assignment operator can also be used to concatenate lists (note one of the original lists will change)



```
list1 = [1, 2, 3]
list2 = [4, 5, 6]
list3 = list1+list2
# list3 now contains [1, 2, 3, 4, 5, 6]

list1 = [1, 2, 3]
list2 = [4, 5, 6]
list1 += list2
#(changes contents of list1 to [1, 2, 3, 4, 5, 6])
```

List Slicing

- Slice: a span of items that are taken from a sequence
  - List slicing format: ***`list[start : end]`***
  - Span is a list containing **copies** of elements from *`start`* up to, but not including, *`end`*
    - If *`start`* not specified, 0 is used for start index
    - If *`end`* not specified, `len(list)` is used for end index
  - Slicing expressions can include a **step** value and negative indexes relative to end of list

```
days = ['Sunday', 'Monday', 'Tuesday', 'Wednesday',
'Thursday', 'Friday', 'Saturday']

midDays = days[2:5]

# Would output ['Tuesday', 'Wednesday', 'Thursday'].
```

Slicing Example

```
days = ['Sunday', 'Monday', 'Tuesday', 'Wednesday',
'Thursday', 'Friday', 'Saturday']

midDays = days[:2]

   # Would output ['Sunday', 'Monday'].
```

Slicing Example

```
days = ['Sunday', 'Monday', 'Tuesday', 'Wednesday',
'Thursday', 'Friday', 'Saturday']

midDays = days[5:]

# Would output ['Friday', 'Saturday '].
```

Slicing Example

```
   days = ['Sunday', 'Monday', 'Tuesday', 'Wednesday',
   'Thursday', 'Friday', 'Saturday']

   midDays = days[2:7:2]

# Would output ['Tuesday', 'Thursday', 'Saturday'].
```

Slicing Example

You can also use negative numbers as indexes to reference positions relative to the end of the list.

```
days = ['Sunday', 'Monday', 'Tuesday', 'Wednesday',
'Thursday', 'Friday', 'Saturday']

midDays = days[-2:]

    # Would output ['Friday', 'Saturday '].
```

How do you think I might change this if I wanted to print out all elements of the list from index -1 back to index 0?

Slicing

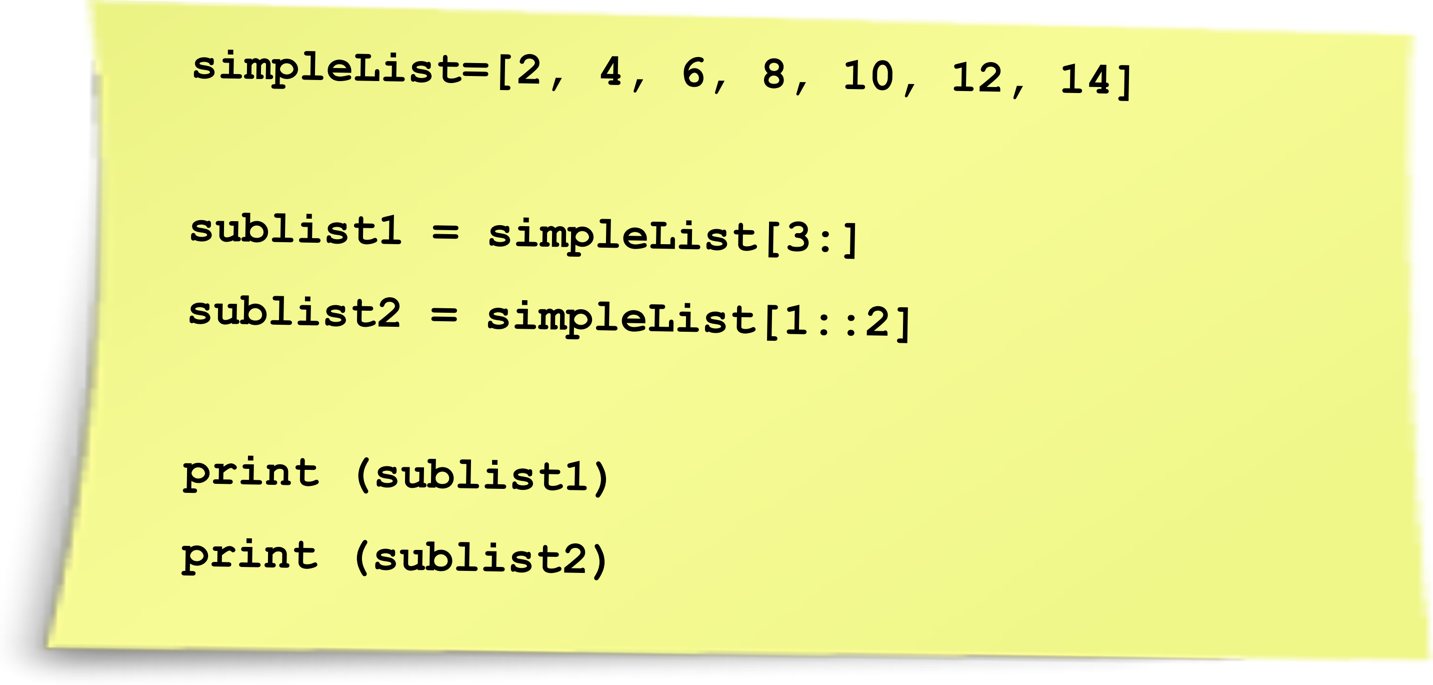
- **Invalid indexes do not** cause slicing expressions to **raise an exception**. For example:

  - If the end index specifies a position beyond the end of the list, Python will use the length of the list instead.

  - If the start index is greater than the end index, the slicing expression will return an empty list.

Task

- Create a list containing the following integer values 2, 4, 6, 8, 10, 12, 14.
  - 1. Use list slicing to return a list containing the values between 8-14 inclusive from the list above
  - 2. Use list slicing to obtain the values 4, 8, 12 from the list above



```
simpleList=[2, 4, 6, 8, 10, 12, 14]


sublist1 = simpleList[3:]
sublist2 = simpleList[1::2]


print (sublist1)
print (sublist2)
```

Task

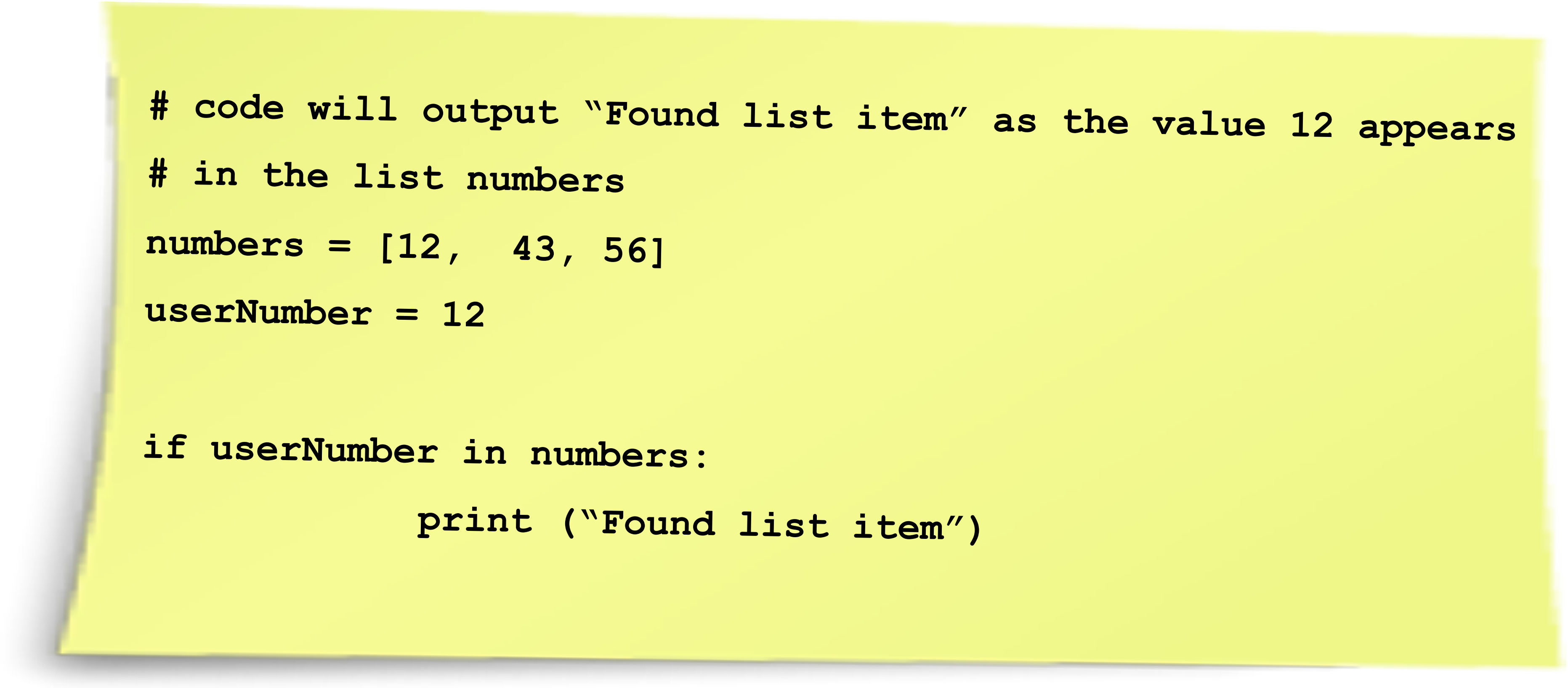
- Write a python program that will generate a list between 0 and 1000
- Use list slicing to print out all numbers in the list that are evenly divisible by 5

```
numbers = list(range(1000))

divisibleNumbers = numbers[5::5]

print (divisibleNumbers)
```

Finding Items in Lists with the `in` Operator

- You can use the `in` operator to determine whether an item is contained in a list
  - General format: ***`item in list`***
  - Returns `True` if the item is in the list, or `False` if it is not in the list
- Similarly you can use the **`not in`** operator to determine whether an item is not in a list

Finding Items in Lists with the `in` Operator



```
# code will output “Found list item” as the value 12 appears
# in the list numbers
numbers = [12,  43, 56]
userNumber = 12

if userNumber in numbers:
        print (“Found list item”)
```

List Methods and Pythons Useful Built-in Functions

- **`append(`*`item`*`)`**: used to add items to a list – *`item`* is appended to the end of the existing list

- **`index(`*`item`*`)`**: used to determine where an item is located in a list
  - Returns the index of the first element in the list containing `item`
  - Raises `ValueError` exception if *`item`* not in the list

- **`insert(`*`index, item`*`)`**: used to insert *`item`* at position *`index`* in the list

Examples (insert)

```
names = ['James', 'Kathryn', 'Bill']

print (names)

# Insert a new name at element 0.
names.insert(0, 'Joe')

print (names)

#The list before the insert:
#['James', 'Kathryn', 'Bill']
#The list after the insert:
#['Joe', 'James', 'Kathryn', 'Bill']
```

Examples (insert and append)

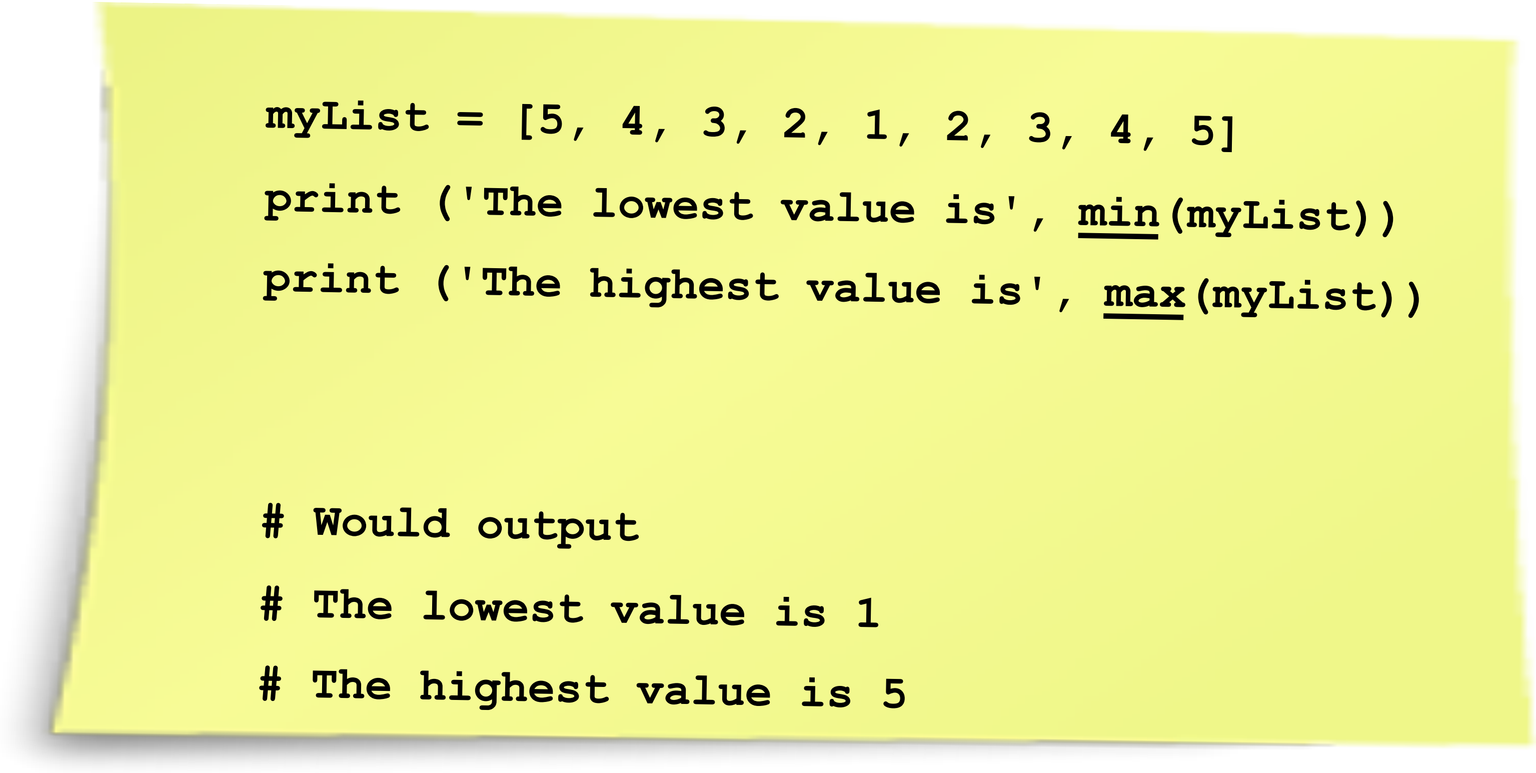
```
days = ['Sunday']
newDay = 'Tuesday'
days.append(newDay)
days.insert(1, 'Monday');
print(days)

# Will output ['Sunday', 'Monday', 'Tuesday']
```

List Methods and Useful Built-in Functions (cont'd.)

- **`del` statement**: removes an element from a specific index in a list
  - General format: `del` *`list`*`[`*`i`*`]`

- **`min` and `max` functions**: built-in functions that return the item that has the lowest or highest value in a sequence
  - The sequence is passed as an argument

Examples (min and max)



```
myList = [5, 4, 3, 2, 1, 2, 3, 4, 5]
print ('The lowest value is', min(myList))
print ('The highest value is', max(myList))


# Would output
# The lowest value is 1
# The highest value is 5
```

Examples (del)

```
myList = [1, 2, 3, 4 , 5]
print ('Before deletion:', myList)
del myList[ 2 ]
print ('After deletion:', myList)



# Would output
# Before deletion: [1, 2, 3, 4, 5]
# After deletion: [1, 2, 4, 5]
```

List Methods and Useful Built-in Functions (cont'd.)

- **`sort()`**: used to sort the elements of the list in ascending order

- **`remove(`*`item`*`)`**: removes the first occurrence of *`item`* in the list

- **`reverse()`**: reverses the order of the elements in the list

Examples (sort, reverse)

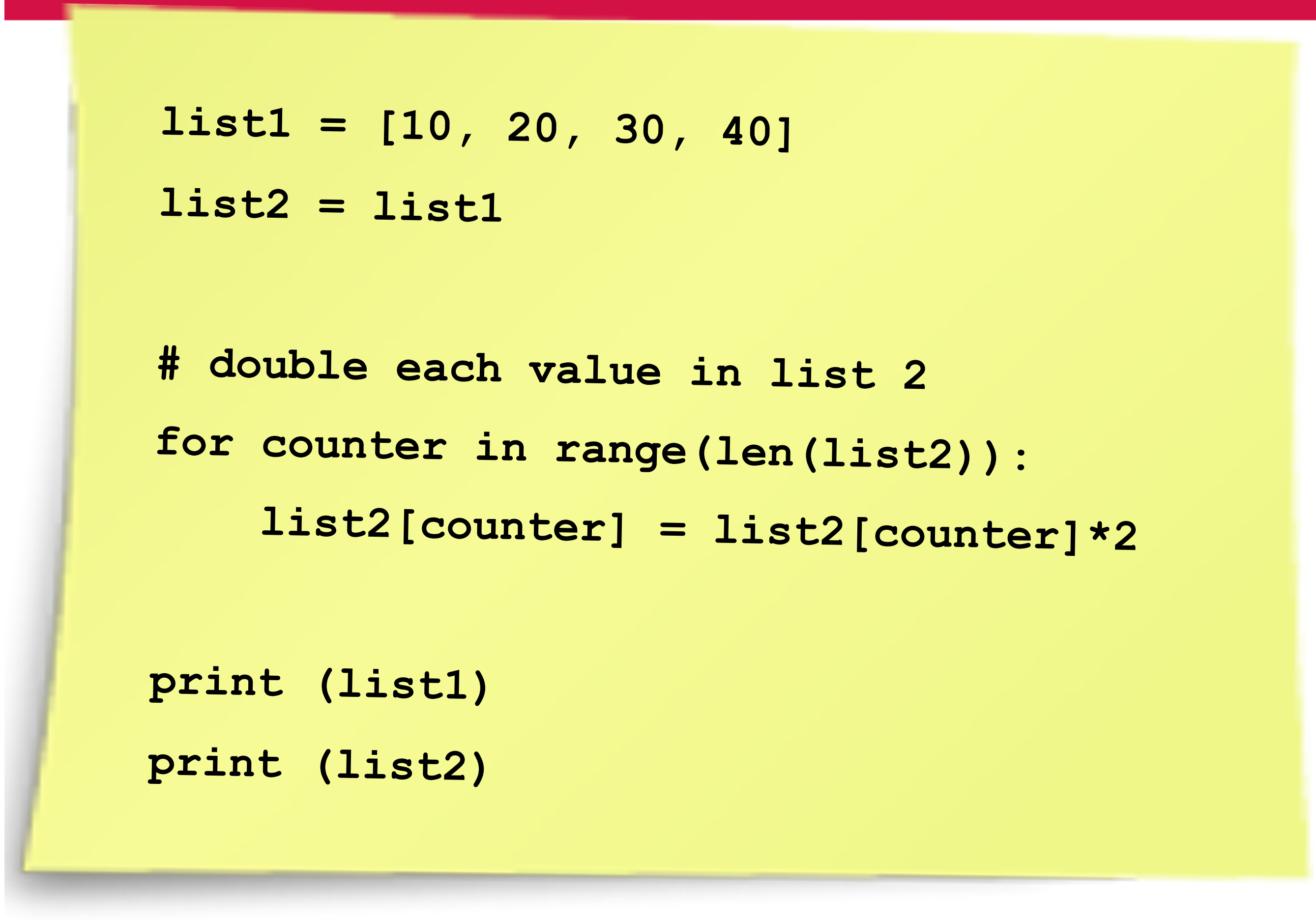
```
myList= [9, 1, 0, 2, 8, 6, 7 , 4, 5]
print ('Original order:', myList)
myList.sort()
print ('Sorted order:', myList)
myList.reverse();
print ('Reverse Sorted order:', myList)

# Original order: [9, 1, 0, 2, 8, 6, 7, 4, 5]
# Sorted order: [0, 1, 2, 4, 5, 6, 7, 8, 9]
# Reverse Sorted order: [9, 8, 7, 6, 5, 4, 2, 1, 0]
```

Exercise

- We want to record the grades of a student in a list.
- We should first ask the user for the total number of subjects and then the individual grades for subject 1, subject 2 etc and store these values in a list
- Next we will loop through the list to obtain the average grade

```
totalNumberSubjects = int(input("How many subjects do you have"))
allNumbers = []

for num in range(totalNumberSubjects):
    num = int(input("Please enter grade for subject"))
    allNumbers.append(num)

print (allNumbers)

total = 0.0
for num in allNumbers:
    total+= num

print ("Average grade is ", total/len(allNumbers))
```

Using Multiple Lists



```
list1 = [10, 20, 30, 40]
list2 = list1

# double each value in list 2
for counter in range(len(list2)):
    list2[counter] = list2[counter]*2

print (list1)
print (list2)
```

What is the output of this program?

Using Multiple Lists

```
list1 = [10, 20, 30, 40]
list2 = list1

# double each value in list 2
for counter in range(len(list2)):
    list2[counter] = list2[counter]*2

print (list1)
print (list2)
```

```
[20, 40, 60, 80]

[20, 40, 60, 80]
```

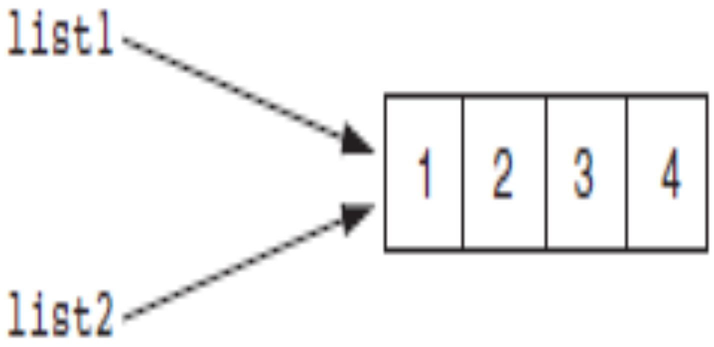
Using Multiple Lists

```
list1 = [10, 20, 30, 40]
list2 = list1

list2[0] = 1500

print (list1)
print (list2)
```

```
[1500, 20, 30, 40]

[1500, 20, 30, 40]
```

Copying Lists

Figure 8-4 list1 and list2 reference the same list



```
# Create a list.
list1 = [1, 2, 3, 4]
# Assign the list to the list2 variable.
list2 = list1
```

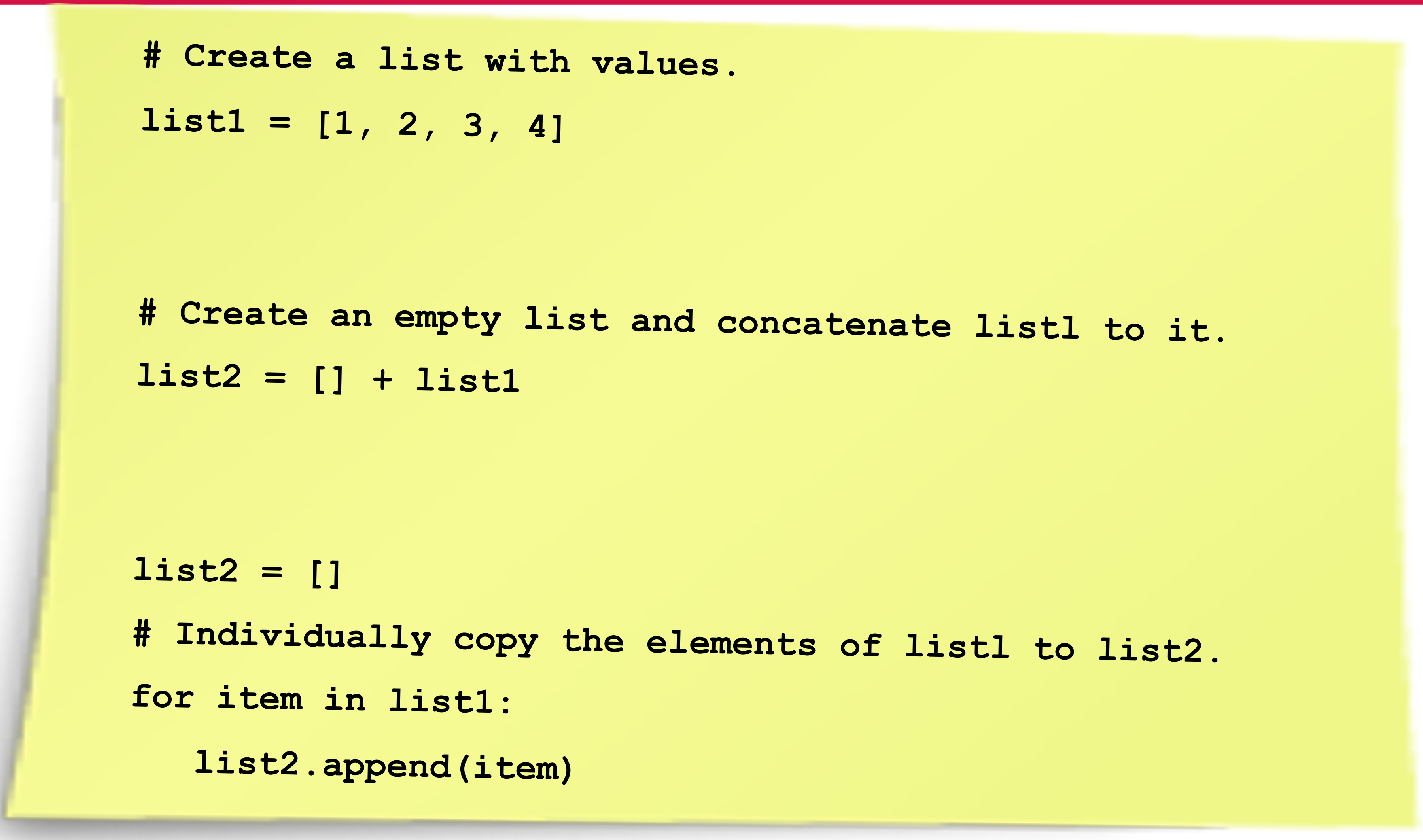
After this code executes, both variables list1 and list2 will reference the same list in memory

Copying Lists

- To make a copy of a list you must copy each element of the list

  - Two methods to do this:
    - Create a new empty list and use a `for` loop to add a copy of each element from the original list to the new list
    - Create a new empty list and concatenate the old list to the new empty list

Examples



```
# Create a list with values.
list1 = [1, 2, 3, 4]


# Create an empty list and concatenate listl to it.
list2 = [] + list1


list2 = []
# Individually copy the elements of listl to list2.
for item in list1:
    list2.append(item)
```

Using Multiple Lists

This code addresses the problems we saw in the previous code and successfully alters the value for only the first element of list2.

```
list1 = [10, 20, 30, 40]
list2 = []+list1

list2[0] = 1500

print (list1)
print (list2)
```

```
[10, 20, 30, 40]

[1500, 20, 30, 40]
```

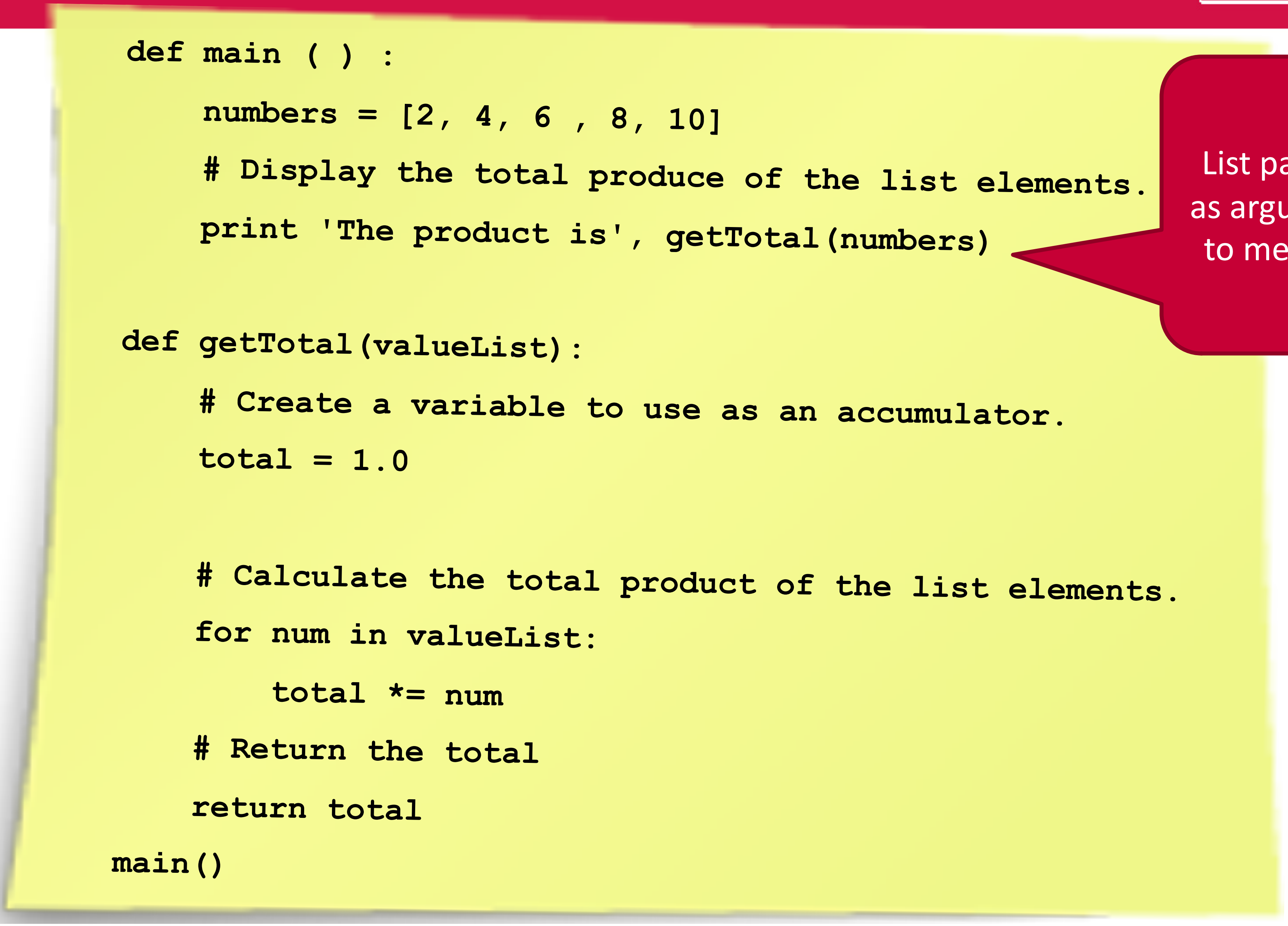
Lists and Functions

- The treatment of lists with functions is similar to the way we treat normal variables with functions.

- The main difference is the impact of changing the values of a list that has been passed into a function as a parameter.

- The example on the following slide shows a basic example of passing a list as an argument to a function.

List as an Argument to a Function



```
def main ( ) :
    numbers = [2, 4, 6 , 8, 10]
    # Display the total produce of the list elements.
    print 'The product is', getTotal(numbers)

def getTotal(valueList):
    # Create a variable to use as an accumulator.
    total = 1.0

    # Calculate the total product of the list elements.
    for num in valueList:
        total *= num
    # Return the total
    return total
main()
```

List passed as argument to method

List Variable is a Mutable Object

```
def main ( ) :
    numbers = [23, 34, 53]
    changeValues(numbers)
    print ('In Main: ',numbers)


def changeValues(allNums):
    allNums[0] = 0
    print ('In changeValue : ',allNums)


main()
```

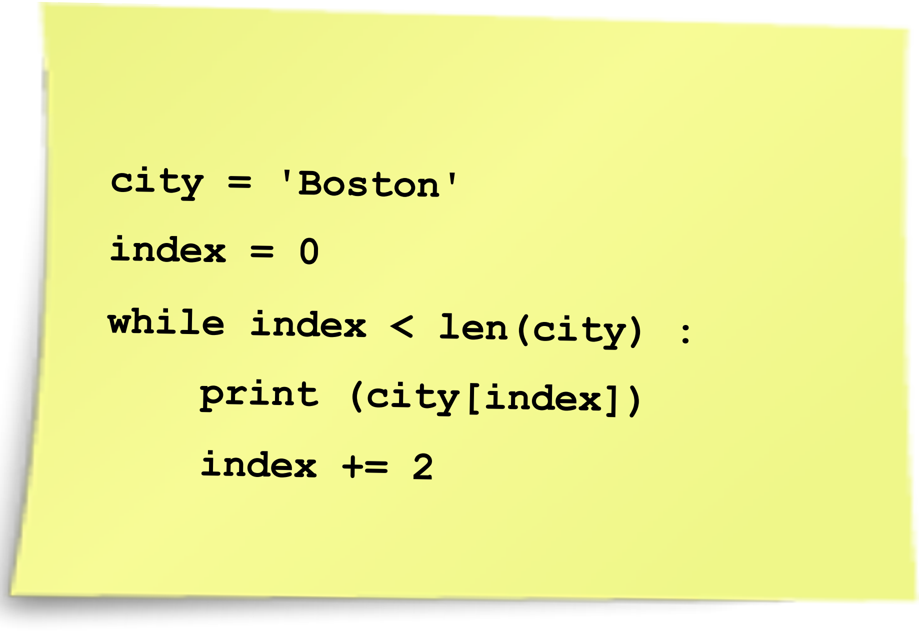
```
In changeValue:  [0, 34, 53]
In Main:  [0, 34, 53]
```

```
def main ( ) :
    number = 5
    changeValue(number)
    print ('In Main: ',number)


def changeValue(num):
    num = 0
    print ('In changeValue: ',num)


main()
```

```
In changeValue :  0
In Main:  5
```

Accessing the Individual Characters in a String

- We can access the individual letters in a word in the same way that we can access elements of a list
- `len(`*`string`*`)` function can be used to obtain the length of a string
  - Useful to prevent loops from iterating beyond the end of a string

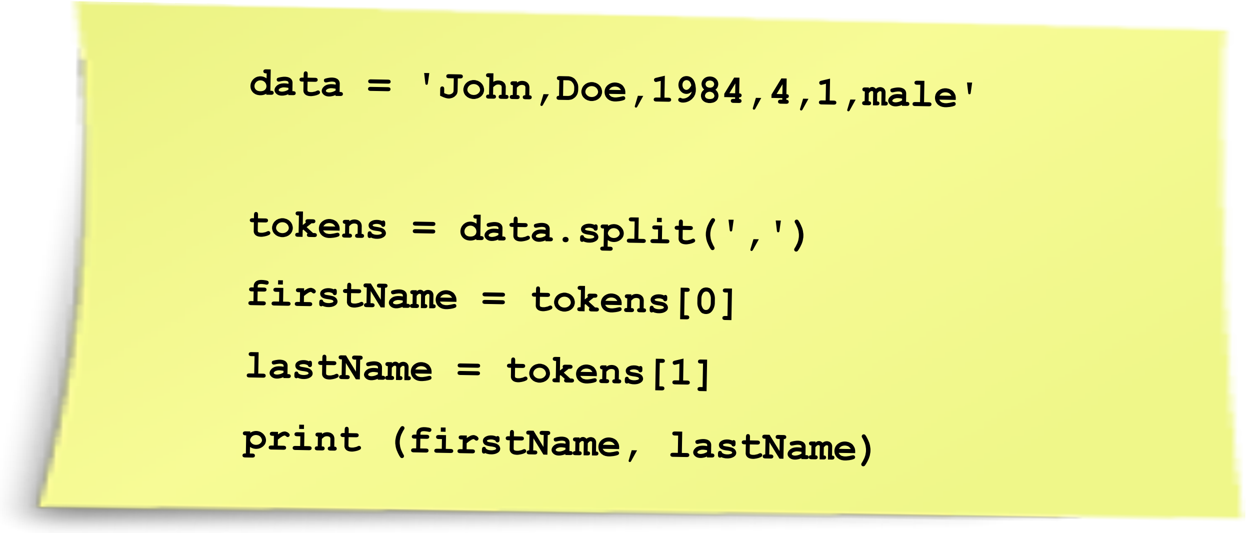


```
city = 'Boston'
index = 0
while index < len(city) :
    print (city[index])
    index += 2
```

Code will print out:
Bso

Split Function

- The split function: returns a list containing the words in the string
  - By default, uses space as separator
  - Can specify a different separator by passing it as an argument to the split method



```
data = 'John,Doe,1984,4,1,male'

tokens = data.split(',')
firstName = tokens[0]
lastName = tokens[1]
print (firstName, lastName)
```

Task

- Ask the user to input a single line sentence from the command line

- Your program should then print out the number of words in the sentence with 3 or less characters

```
def main ( ) :
    text = input('Please input a sentence')
    words = text.split()

    count = 0

    for word in words:
        if len(word)<=3:
            count+=1

    print ('Total number of words with 3 or less chars', count)
```

Topics

- Lists
- Dictionaries
- Sets

Dictionaries

- Dictionary: object that stores a collection of data
  - Each element consists of a *key* and a *value* pair
  - To retrieve a specific value, use the key associated with it
  - Format for creating a dictionary

    ```
    dictionary =  {key1:val1, key2:val2}
    ```

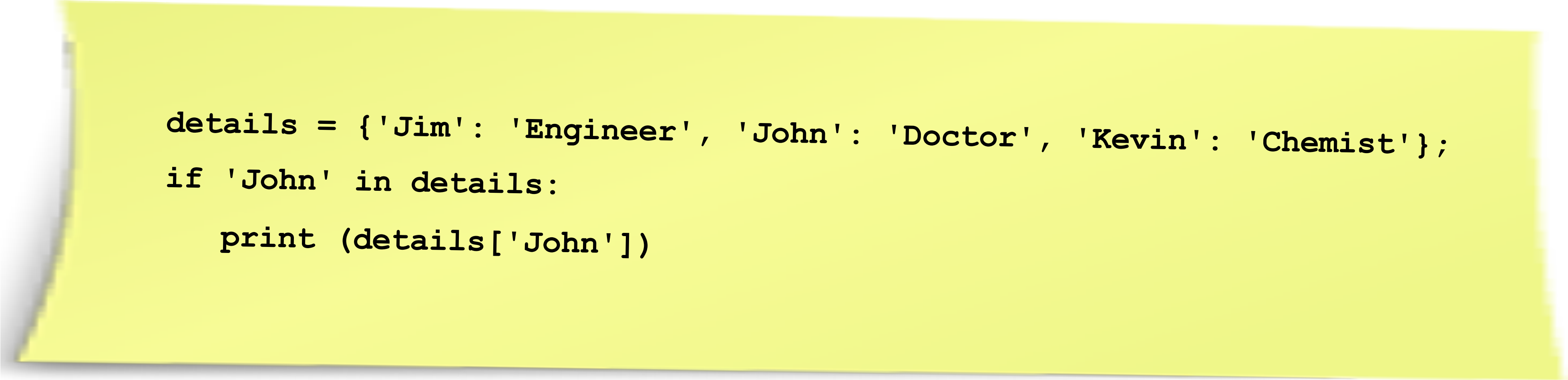
  - To create an empty dictionary:

    ```
    Use dictionary ={}
    ```

```
details = {‘Jim': ‘Engineer', ‘John': ‘Doctor’, ‘Kevin': ‘Chemist'}
```

Retrieving a Value from a Dictionary

- Elements in a dictionary are unsorted
- General format for retrieving a value from dictionary:
  `dictionary[key]`
- Test whether a key is in a dictionary using the `in` and `not in` operators



```
details = {'Jim': 'Engineer', 'John': 'Doctor', 'Kevin': 'Chemist'};
if 'John' in details:
   print (details['John'])
```

Adding Elements to an Existing Dictionary

- To add a new key-value pair:

  ```
  dictionary[key] = value
  ```

  - If key exists in the dictionary, the value associated with it will be changed
  - If key doesn't exist in the dictionary, then a new key, value pair is added to the dictionary

- len function: used to obtain number of elements in a dictionary

  ```
  len(dictionary)
  ```

Dictionaries (Adding key value pair to a dictionary)

Adds the key value pair ‘John’:’Teacher’ to the dictionary

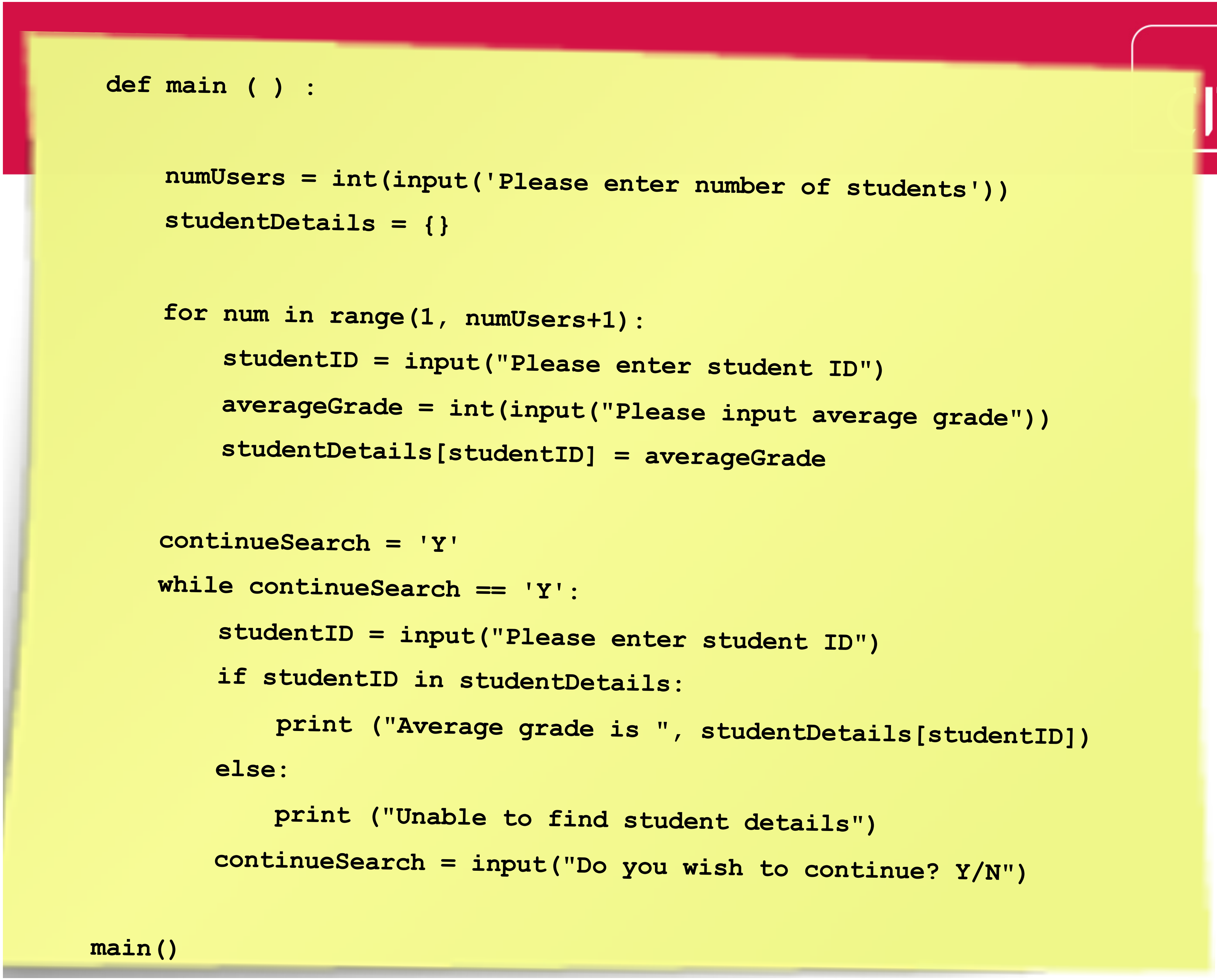
```
details = {Jim': ‘Engineer', ‘John': ‘Doctor’, ‘Kevin': ‘Chemist'};

details[‘John']= ‘Teacher’;
```

{'Jim': 'Engineer', 'John': 'Teacher', 'Kevin': 'Chemist'}

Task

- Create a program that will ask the user to enter the average grades attained by each student and store this data in a dictionary.
  - The program should first ask for the number of students.
  - For each student it should ask for the student number and average numerical grade.

- When finished entering data the user should be able to query the dictionary by entering a student ID.
  - If the student exists is the dictionary their average grade should be printed
  - If not then an error message should be printed.

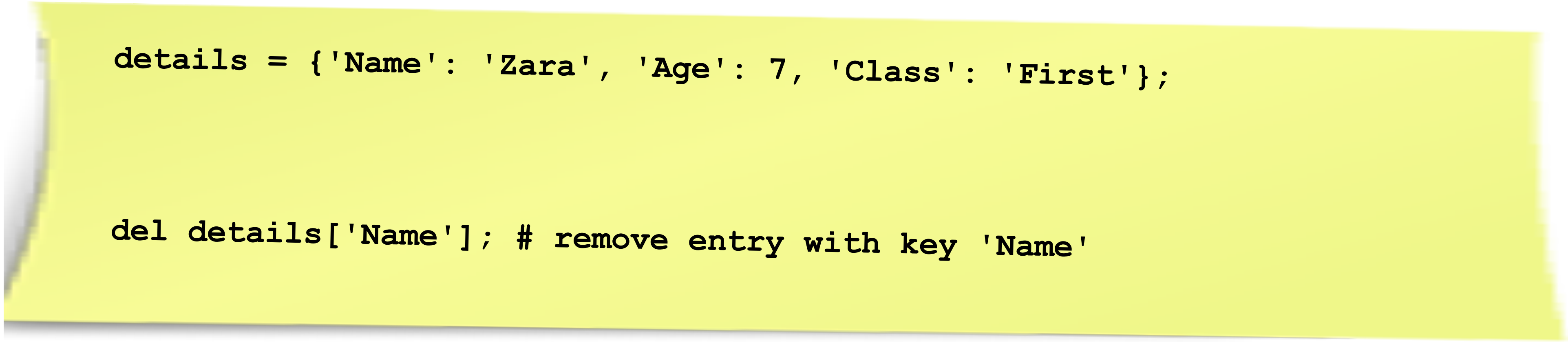


```
def main ( ) :

    numUsers = int(input('Please enter number of students'))
    studentDetails = {}

    for num in range(1, numUsers+1):
        studentID = input("Please enter student ID")
        averageGrade = int(input("Please input average grade"))
        studentDetails[studentID] = averageGrade

    continueSearch = 'Y'
    while continueSearch == 'Y':
        studentID = input("Please enter student ID")
        if studentID in studentDetails:
            print ("Average grade is ", studentDetails[studentID])
        else:
            print ("Unable to find student details")
        continueSearch = input("Do you wish to continue? Y/N")

main()
```

Deleting Elements From an Existing Dictionary

- To delete a key-value pair:

```
del dictionary[key]
```



```
details = {'Name': 'Zara', 'Age': 7, 'Class': 'First'};


del details['Name']; # remove entry with key 'Name'
```

Removes the key value pair ‘Name’:’Zara’ from the dictionary details.
Again you should use the in operator to determine if the key/value pair is present

Dictionaries (Delete element from a dictionary)

```
details = {'Name': 'Zara', 'Age': 7, 'Class': 'First'};

if 'Name' in details:
    del details['Name']; # remove entry with key 'Name'
```

Inclusion of test to determine if Name is a valid key within the dictionary. Avoid raising an exception

Properties of Dictionary Keys

- **Dictionary values have no restrictions**. They can be any arbitrary Python objects, either standard objects or user-defined objects.

- However, same is not true for the keys.

- There are two important points to remember about dictionary keys:
  - More than one entry per key not allowed. Which means **no duplicate key** is allowed. When duplicate keys encountered during assignment, the last assignment wins.
  - **Keys must be immutable**. Which means you can use strings and numbers as dictionary keys.
  - https://docs.python.org/2/reference/datamodel.html

Dictionaries (Allowable Key Types)

```
dict = {['Name']: 'Zara', 'Age': 7}
```

The key provided above is a mutable type. It is a list.

The interpreter will raise an error Indicating that a list is an unhashable type

Dictionaries (Using a list as the value element in a dictionary and the len method)

```
testScore = {'Scully':[56, 67, 34,],
                'Fitzgibbon':[78, 76, 54]}
print (testScore['Scully'])
print (len(testScore))
```

Will print out:

```
[56, 67, 34]
2
```

Dictionaries (Using a for loop)

Use a for loop to iterate over a dictionary

General format: *for key in dictionary:*

```
ages = {}

#Add a couple of names to the dictionary
ages['Sue'] = 23
ages['Peter'] = 19
ages['Andrew'] = 78
ages['Karren'] = 45

for key in ages:
    print (ages[key])
```

Use a for loop to iterate through the keys in the dictionary

```
23
19
78
45
```

Some Dictionary Methods

- `clear` method: deletes all the elements in a dictionary, leaving it empty
  - Format: *`dictionary`*`.clear()`

- `keys` method: returns all the dictionary's keys as a list
  - Format: *`dictionary`*`.keys()`

- `values` method: returns all the values in the dictionary as a list
  - Format: *`dictionary`*`.values()`
  - Use a `for` loop to iterate over the values

Dictionaries (Using the keys method)

```
ages = {}

#Add a couple of names to the dictionary
ages['Sue'] = 23
ages['Peter'] = 19
ages['Andrew'] = 78
ages['Karren'] = 45

print ("The following people are in the dictionary:")
print (ages.keys())
```

Program produces the following output

```
The following people are in the dictionary:
['Sue', 'Peter', 'Andrew', 'Karren']
```

Using Dictionaries with Methods

- A dictionary is a mutable data structure and as such it behaves in the same way as a list when used with functions.

- In other words any change made to a dictionary parameter in a function is reflected in the original argument.

Using Dictionaries with Methods

```
def main():
    ages = {}
    ages['Sue'] = 23
    addEntry(ages)
    print (ages)

def addEntry(ages):
    ages['John'] = 13

main()
```

{'Sue': 23, 'John': 13}

Notice the change made in the function to the parameter is reflected in the original argument

Topics

- Lists
- Dictionaries
- ➡ Sets

Sets

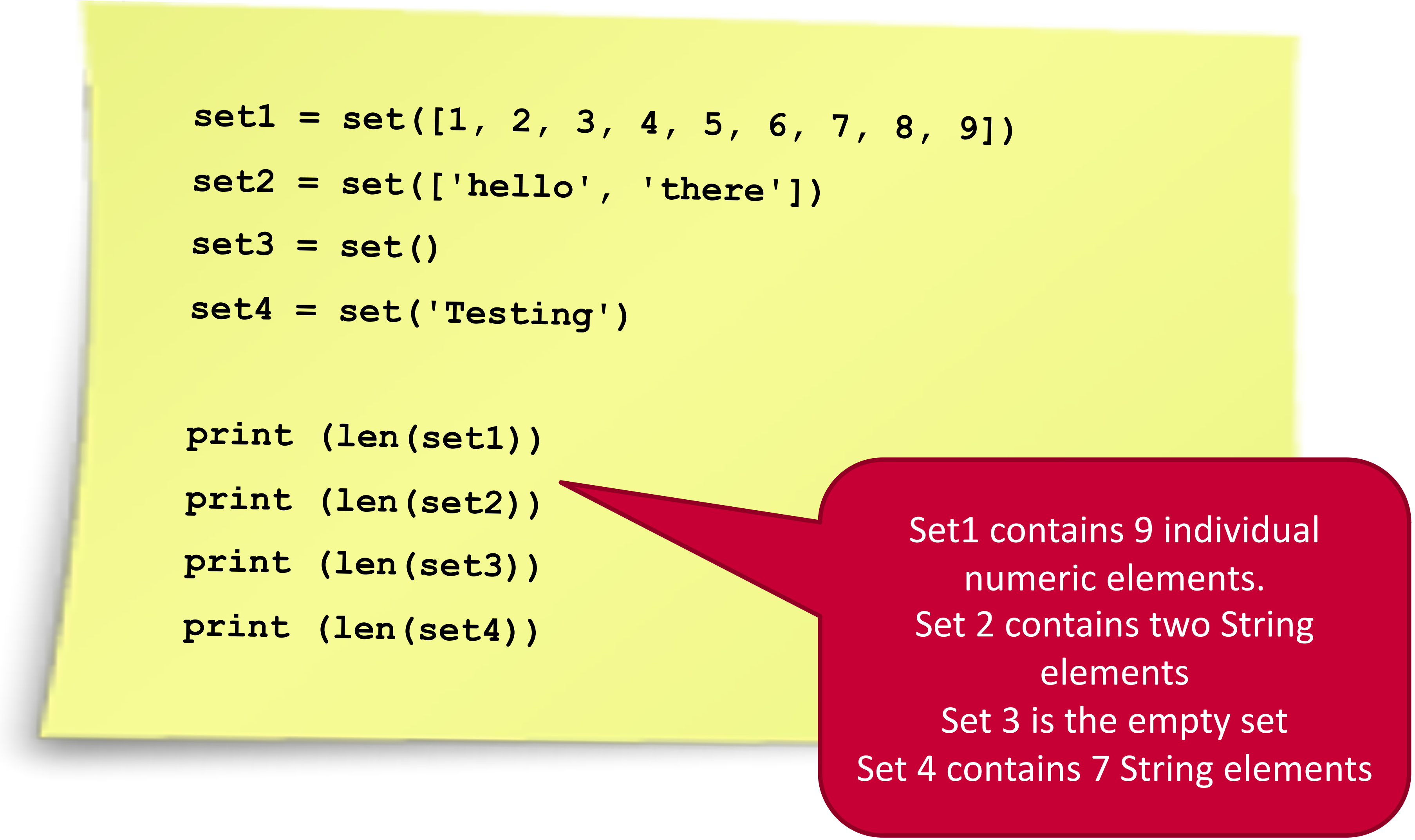
- Set: object that stores a collection of data in the same way as a mathematical set.
  - All items must be **unique**. No two elements can have the same value.
  - Set is **unordered**
  - Elements can be of different data types

Creating a Set

- `set` function: used to create a set
  - For empty set, call `set()`
  - For non-empty set, call `set(argument)` where *`argument`* is an object that contains iterable elements (list or string)
    - e.g. *`argument`* can be a list
    - If *`argument`* contains duplicates, only one of the duplicates will appear in the set

- `len` function: returns the number of elements in the set

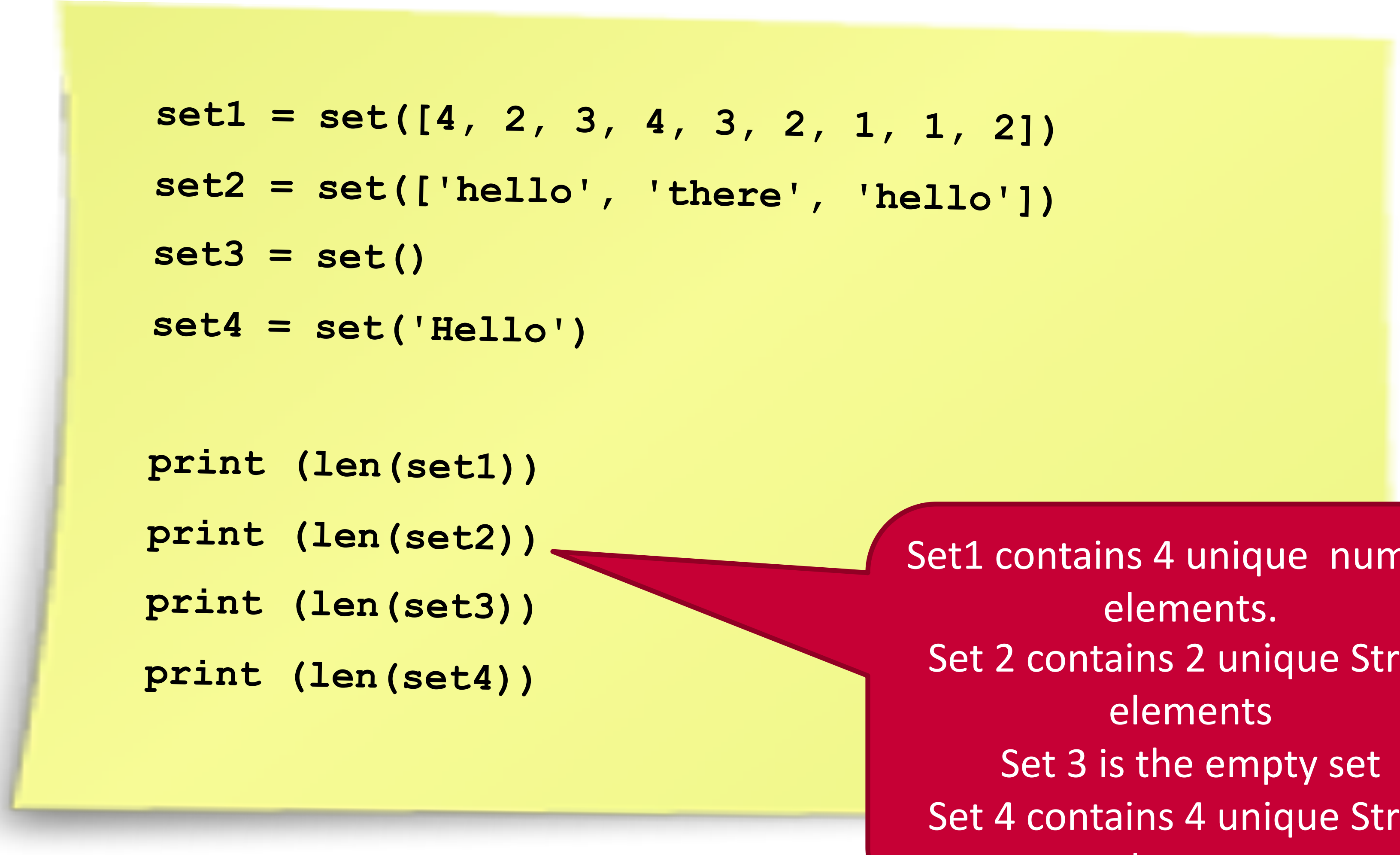
Sets (Creation)



```
set1 = set([1, 2, 3, 4, 5, 6, 7, 8, 9])
set2 = set(['hello', 'there'])
set3 = set()
set4 = set('Testing')

print (len(set1))
print (len(set2))
print (len(set3))
print (len(set4))
```

Set1 contains 9 individual numeric elements.
Set 2 contains two String elements
Set 3 is the empty set
Set 4 contains 7 String elements

Sets (Creation)



```
set1 = set([4, 2, 3, 4, 3, 2, 1, 1, 2])
set2 = set(['hello', 'there', 'hello'])
set3 = set()
set4 = set('Hello')

print (len(set1))
print (len(set2))
print (len(set3))
print (len(set4))
```

Set1 contains 4 unique numeric elements.
Set 2 contains 2 unique String elements
Set 3 is the empty set
Set 4 contains 4 unique String elements

Sets (Creation)

- Sets don’t allow duplicates.

- What if we want to create a set in which each element is a string containing more than one character?

- For example how would you create a set containing the elements ‘one’, ‘two’ and ‘three’?
- Would either of the two options below suffice?

```
set1 = set(‘one’, ‘two’, ‘three’)
set2 =  set(‘one two three’)
```

First method is illegal as the set method will only accept a single argument
Second method will produce a set consisting of just individual characters

```
# Create set using a list
set1 = set([‘one’, ‘two’, ‘three’])
```

Getting the Number of and Adding Elements

- Sets are mutable objects
  - **`add` method**: adds an element to a set
  - **`update` method**: adds a group of elements to a set
    - Single argument must be iterable element, and each of the elements is added to the set

Sets (Add, Update method)

What is the output of the code below?

```
set1 = set()
set1.add(1)
set1.update([2, 10])
print (set1)
print (len(set1))
```

Program outputs the following:

```
{1, 2, 10}
3
```

Sets (Add, Update method)

What is the output of the code below?

```
set1 = set()
set1.add("Hello")
set1.update("There")
print (set1)
```

Program outputs the following:

{'h', 'T', 'r', 'Hello', 'e'}

Sets (Add, Update method)

Update can also accept another set as an argument

```
set1 = set([1,2, 3])
set2 =set([3, 4, 5, 6])
set1.update(set2)
print (set1)
```

Prints out:
{1, 2, 3, 4, 5, 6}

Deleting Elements From a Set

- `remove` : remove the specified item from the set
  - The item that should be removed is passed to method as an argument

- `clear` method: clears all the elements of the set

- The `in` operator can be used to test whether a value exists in a set
  - Similarly, the `not in` operator can be used to test whether a value does not exist in a set

Sets (remove method)

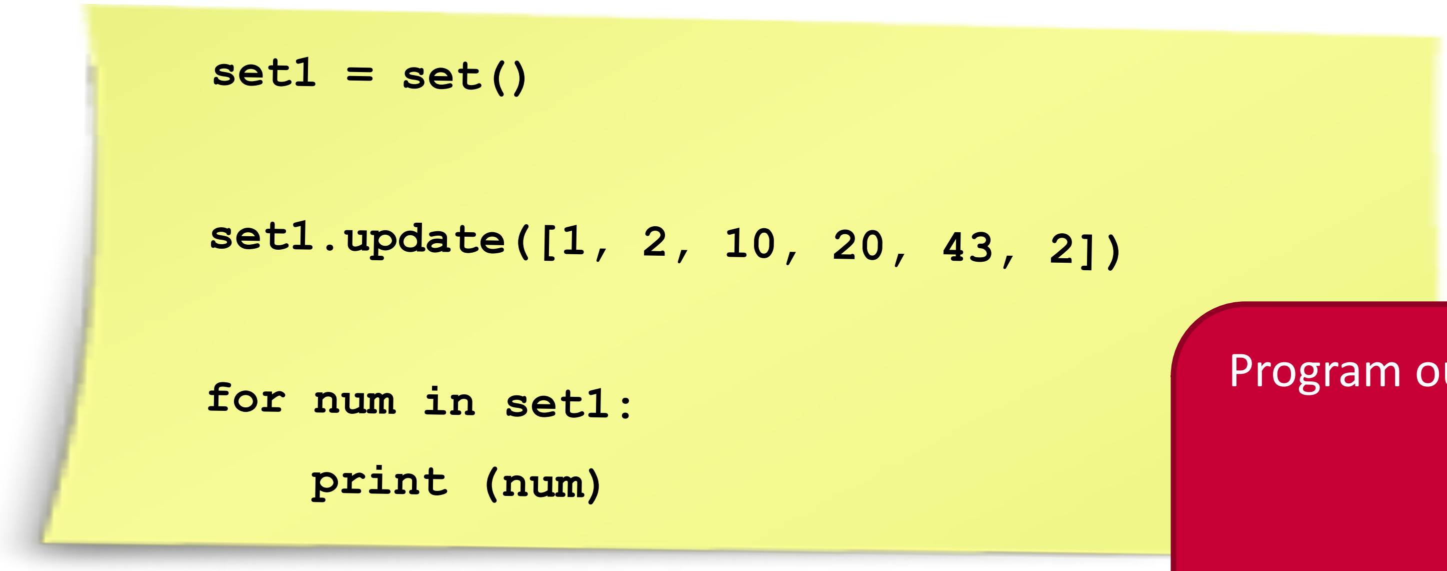
```
set1 = set()

set1.update([1, 2, 10, 20])
print (set1)

num = 1
if num in set1:
    set1.remove(num)

print (set1)
```

Program outputs the following:

```
{1, 2, 10, 20}
{2, 10, 20}
```

Using the `for` Loop With a Set

- A `for` loop can be used to iterate over elements in a set
  - General format: `for` *`item`* `in` *`set`*`:`
  - The loop iterates once for each element in the set



```
set1 = set()

set1.update([1, 2, 10, 20, 43, 2])

for num in set1:
    print (num)
```

Program outputs the following:

```
1
2
10
43
20
```

Finding the Union of Sets

- To find the union of two sets:
  - Use the `union` method
    - Format: `set1.union(set2)  or set1 | set2`
    - Returns a new set which contains the union of both sets

- Intersection of two sets: a set that contains only the elements found in both sets
  - Use the intersection method
  - Format: set1.intersection(set2) or set1 & set2

Sets (Intersection and Union)

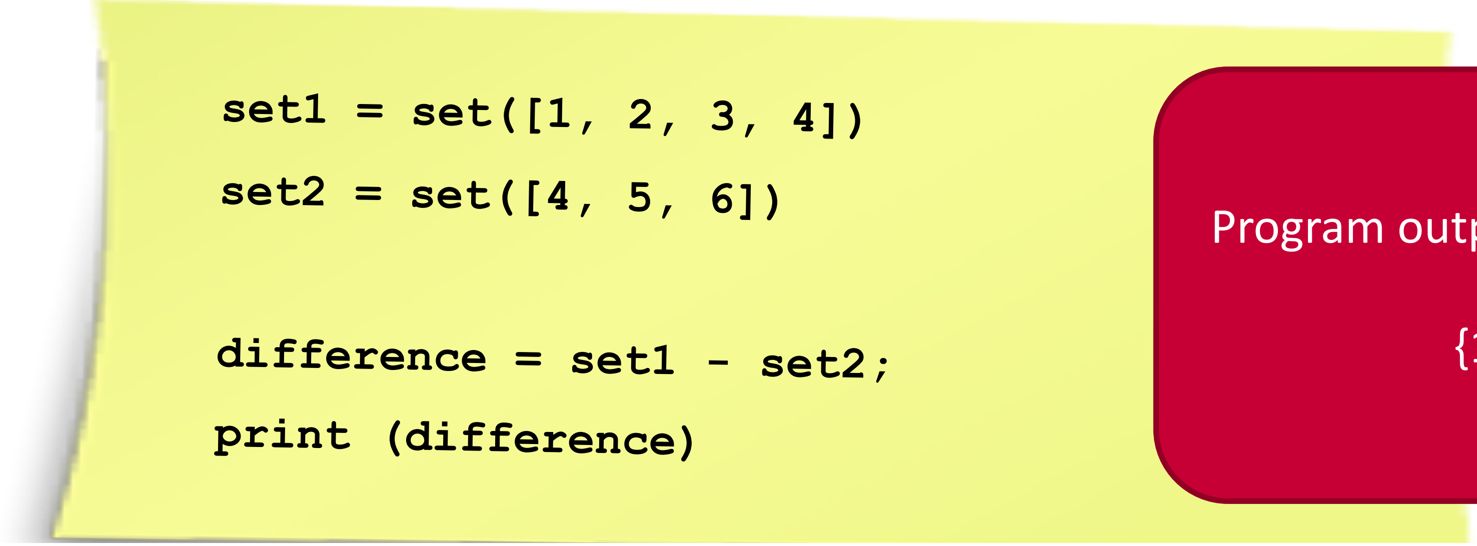
```
set1 = set([1, 2, 3, 4])
set2 = set([4, 5, 6])
set3 = set([4, 7, 8, 9])

set4 = set1 | set2 | set3
print (set4)

set5 = set1 & set2 & set3
print (set5)
```

Program outputs the following:

```
{1, 2, 3, 4, 5, 6, 7, 8, 9}
{4}
```

Finding the Difference of Sets

- Difference of two sets: a set that contains the elements that appear in the first set but do not appear in the second set
- To find the difference of two sets:
  - Use the `difference` method
    - Format: *`set1`*`.difference(`*`set2`*`)`
  - Use the − operator
    - Format: *`set1`*` - `*`set2`*



```
set1 = set([1, 2, 3, 4])
set2 = set([4, 5, 6])

difference = set1 - set2;
print (difference)
```

Program outputs the following:

{1, 2, 3}

Finding Subsets and Supersets

- To determine whether set A is subset of set B
  - Use the `issubset` method
    - Format: `setA.issubset(setB)`
  - Use the <= operator
    - Format: `setA <= setB`

- To determine whether set A is superset of set B
  - Use the `issuperset` method
    - Format: `setA.issuperset(setB)`
  - Use the >= operator
    - Format: `setA >= setB`

Sets (Intersection and Union)

```
set1 = set([1, 2, 3, 4])
set2 = set([4, 2])

result1 = set2 <= set1
print (result1)

result2 = set2 >= set1
print (result2)
```

The first print statement will print **true** because set2 is a subset of set 1.
The second print statement will print **false** because set 2 is not a superset of set1.

Programming Task

- Write a program that will initially gather information on employees and their salaries. The program should continue asking the user for an employee **surname** and **salary** until the user wishes to finish. This data should be stored in a dictionary.
- Next the program should print the total salary bill for the company (using only the dictionary)
- It should then print out the list of unique salaries of all employees.

```
def main():
    employeeDetails = {}
    continueEnteringData = 'y'

    while continueEnteringData == 'y':
        surname = input('Please input employee surname')
        salary = int(input('Please input salary'))
        employeeDetails[surname] = salary

        continueEnteringData = input('Enter details for another employee?')

    totalSalaryBill = 0

    for key in employeeDetails:
        totalSalaryBill += employeeDetails[key]
    print ("Total Salary: ", totalSalaryBill)

    uniqueNames = set(employeeDetails.values())
    print (uniqueNames)
```

Finding the Symmetric Difference of Sets

- Symmetric difference of two sets: a set that contains the elements that are not shared by the two sets
- To find the symmetric difference of two sets:
  - Use the `symmetric_difference` method
    - Format: `set1.symmetric_difference(set2)`
  - Use the ^ operator
    - Format: `set1 ^ set2`

```
set1 = set([1, 2, 3, 4])
set2 = set([4, 5, 6])

difference = set1 ^ set2;
print difference
```

Program outputs the following:

```
set([1, 2, 3, 5, 6])
```

Some Dictionary Methods

- `clear` method: deletes all the elements in a dictionary, leaving it empty
  - Format: `dictionary.clear()`

- `get` method: gets a value associated with specified key from the dictionary
  - Format: `dictionary.get(key, default)`
    - *`default`* is returned if *`key`* is not found
  - Alternative to `[]` operator
    - Cannot raise `KeyError` exception

Dictionaries (Using a get method)

```
ages = {}

#Add a couple of names to the dictionary
ages['Sue'] = 23
ages['Peter'] = 19
ages['Andrew'] = 78
ages['Karren'] = 45

value = ages.get(‘John’, ‘Entry not found’)
print value
```

Because the key ‘John’ does appear in the dictionary the string ‘Entry not found’ will be returned and stored in the value variable

Some Dictionary Methods (cont'd.)

- `keys` **method**: returns all the dictionaries keys as a list
  - Format: *`dictionary`*`.keys()`
- `pop` **method**: returns value associated with specified key and removes that key-value pair from the dictionary
  - Format: *`dictionary`*`.pop(`*`key, default`*`)`
    - *`default`* is returned if *`key`* is not found
- `values` **method**: returns all the values in the dictionary as a list
  - Format: *`dictionary`*`.values()`
  - Use a `for` loop to iterate over the values

Dictionaries (Using the keys method)

```
ages = {}

#Add a couple of names to the dictionary
ages['Sue'] = 23
ages['Peter'] = 19
ages['Andrew'] = 78
ages['Karren'] = 45

print "The following people are in the dictionary:"
print ages.keys()
```

Program produces the following output

```
The following people are in the dictionary:
['Sue', 'Peter', 'Andrew', 'Karren']
```