Python

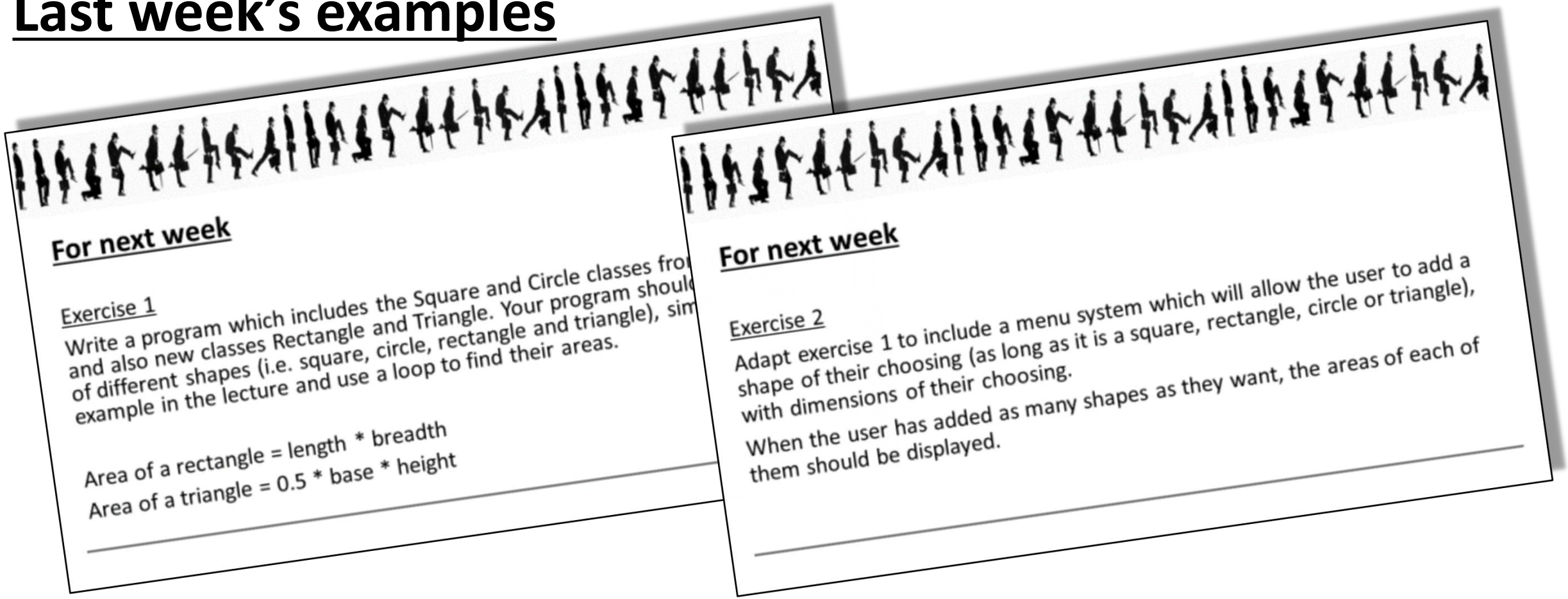
Introduction to Programming

Comp07027

Lecture 13

More Data Structures and More OOP

Last week's examples



For next week

Exercise 1

Write a program which includes the Square and Circle classes fro
and also new classes Rectangle and Triangle. Your program shoul
of different shapes (i.e. square, circle, rectangle and triangle), sim
example in the lecture and use a loop to find their areas.

Area of a rectangle = length * breadth
Area of a triangle = 0.5 * base * height

For next week

Exercise 2

Adapt exercise 1 to include a menu system which will allow the user to add a shape of their choosing (as long as it is a square, rectangle, circle or triangle), with dimensions of their choosing.

When the user has added as many shapes as they want, the areas of each of them should be displayed.

Last week's examples

Let's look at a possible partial solution to the problem.

Last week's examples

Square class with a couple of additions

```
class Square:
    shape_name = "square   "

    def __init__(self, side):
        self.side = side

    def calc_area(self):
        return self.side ** 2

    def calc_circum(self):
        return self.side * 4
```

Last week's examples

Square class with a couple of additions

```
class Square:
    shape_name = "square   "

    def __init__(self, side):
        self.side = side

    def calc_area(self):
        return self.side ** 2

    def calc_circum(self):
        return self.side * 4
```

If we have a value which is EXACTLY the same for every instance of Square (in this case the name "square") we can declare it as a constant outside __init__

Last week's examples

Square class with a couple of additions

```
class Square:
    shape_name = "square   "

    def __init__(self, side):
        self.side = side

    def calc_area(self):
        return self.side ** 2

    def calc_circum(self):
        return self.side * 4
```

If we have a value which is EXACTLY the same for every instance of Square (in this case the name "square") we can declare it as a constant outside __init__

The method to calculate the area of the square.

Last week's examples

Square class with a couple of additions

```
class Square:
    shape_name = "square   "

    def __init__(self, side):
        self.side = side

    def calc_area(self):
        return self.side ** 2

    def calc_circum(self):
        return self.side * 4
```

If we have a value which is EXACTLY the same for every instance of Square (in this case the name "square") we can declare it as a constant outside __init__

The method to calculate the area of the square.

We can add more than one method – here I've added a method to calculate the circumference of the square.

Last week's examples

Rectangle class with a couple of additions

```
class Rectangle:
    shape_name = "rectangle"

    def __init__(self, length, breadth):
        self.length = length
        self.breadth = breadth

    def calc_area(self):
        return self.length * self.breadth

    def calc_circum(self):
        return (self.length * 2) + (self.breadth * 2)
```

Last week's examples

Rectangle class with a couple of additions

```
class Rectangle:
    shape_name = "rectangle"

    def __init__(self, length, breadth):
        self.length = length
        self.breadth = breadth

    def calc_area(self):
        return self.length * self.breadth

    def calc_circum(self):
        return (self.length * 2) + (self.breadth * 2)
```

Rectangle is similar to Square.
I've added the constant shape_name

Last week's examples

Rectangle class with a couple of additions

```
class Rectangle:
    shape_name = "rectangle"

    def __init__(self, length, breadth):
        self.length = length
        self.breadth = breadth

    def calc_area(self):
        return self.length * self.breadth

    def calc_circum(self):
        return (self.length * 2) + (self.breadth * 2)
```

Rectangle is similar to Square.
I've added the constant shape_name

The method to calculate the area of the rectangle is similar to square except that it requires TWO dimensions.

Last week's examples

Rectangle class with a couple of additions

```
class Rectangle:
    shape_name = "rectangle"

    def __init__(self, length, breadth):
        self.length = length
        self.breadth = breadth

    def calc_area(self):
        return self.length * self.breadth

    def calc_circum(self):
        return (self.length * 2) + (self.breadth * 2)
```

Rectangle is similar to Square.
I've added the constant shape_name

The method to calculate the area of the rectangle is similar to square except that it requires TWO dimensions.

Calc_circum() also needs two inputs.

Last week's examples

Here's the rest of the program.

Can you see what it does?

```
list_of_shapes = []

def add_square(s):
    list_of_shapes.append(Square(s))

def add_rectangle(l, b):
    list_of_shapes.append(Rectangle(l, b))

def menu():
    print("Enter 1 to add a SQUARE")
    print("Enter 2 to add a RECTANGLE")
    menu_choice = input("> ")
    if menu_choice == "1":
        sd = int(input("Enter side > "))
        add_square(sd)
    elif menu_choice == "2":
        leng = int(input("Enter length > "))
        bread = int(input("Enter breadth > "))
        add_rectangle(leng, bread)
    else:
        print("Invalid choice")

quit_menu = False

while quit_menu is False:
    menu()
    another_shape = input("Another shape?  Y or N")
    if (another_shape == "N") or (another_shape == "n"):
        quit_menu = True
        for shape in list_of_shapes:
            print(shape.shape_name, "\t\tarea = ", shape.calc_area(), "\t\tcircumference = ", shape.calc_circum())
        print("All done")
```

Last week's examples

Here's the output from the program.

This is not a definite solution to the problem but might help you see how to use OOP.

```
"C:\Program Files\Python36\python.exe" "F:/IntroToPython 19-20/tuples dictionaries/sq_cir_rect_tri.py"
Enter 1 to add a SQUARE
Enter 2 to add a RECTANGLE
> 1
Enter side > 1
Another shape?  Y or NY
Enter 1 to add a SQUARE
Enter 2 to add a RECTANGLE
> 2
Enter length > 2
Enter breadth > 3
Another shape?  Y or NY
Enter 1 to add a SQUARE
Enter 2 to add a RECTANGLE
> 1
Enter side > 3
Another shape?  Y or NY
Enter 1 to add a SQUARE
Enter 2 to add a RECTANGLE
> 2
Enter length > 5
Enter breadth > 7
Another shape?  Y or NN
square          area =  1       circumference =  4
rectangle       area =  6       circumference =  10
square          area =  9       circumference =  12
rectangle       area =  35      circumference =  24
All done

Process finished with exit code 0
```

More on Data Structures

Data Structures

So far we have made great use of **LISTS** in python (as we did in the assessment and the previous example).

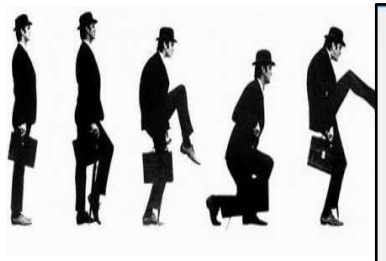
However, there are others and we're now going to have a look at a couple of them – **TUPLES** and **DICTIONARIES**.

Tuple

A **Tuple** is like a List.

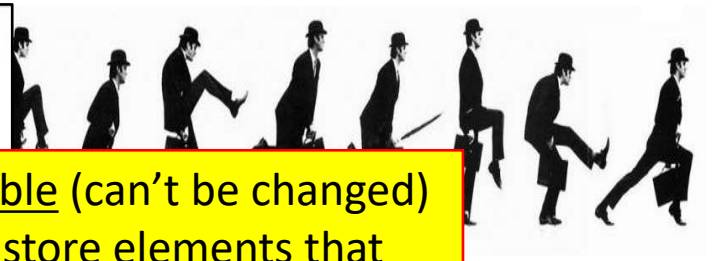
The difference is that a tuple cannot be changed, so we tend to use tuples when we want to create a list of elements that are set. Tuples takes less time to process.

Tuples are said to be **immutable** (can’t be changed)

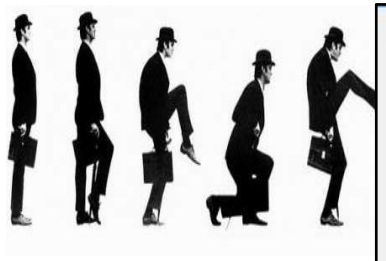


Tuple

```
week_days = ("Monday", "Tuesday", "Wednesday", "Thursday", "Friday")
weekends = ("Saturday", "Sunday")

#we can create a new TUPLE from existing ones
whole_week = week_days + weekends
print()
print("whole_week =", whole_week)



#we can isolate an element
print()
print("The 4th element of whole_week is", whole_week[3])

#we can slice a TUPLE
mid_week = week_days[1:4]
print()
print("A [1:4] slice of whole_week is", mid_week)

#we can iterate through a TUPLE
print()
for day in week_days:
    print(day)
```

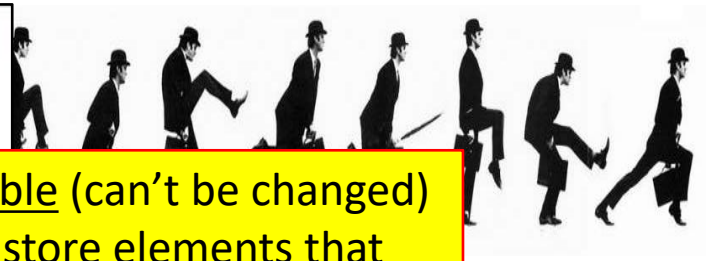


Tuples are <u>immutable</u> (can't be changed) so we use them to store elements that won't change – like the days of the week, the colours of the rainbow etc



Tuple

```
week_days = ("Monday", "Tuesday", "Wednesday", "Thursday", "Friday")
weekends = ("Saturday", "Sunday")

#we can create a new TUPLE from existing ones
whole_week = week_days + weekends
print()
print("whole_week =", whole_week)


#we can isolate an element
print()
print("The 4th element of whole_week is", whole_week[3])

#we can slice a TUPLE
mid_week = week_days[1:4]
print()
print("A [1:4] slice of whole_week is", mid_week)

#we can iterate through a TUPLE
print()
for day in week_days:
    print(day)
```



Tuples are <u>immutable</u> (can't be changed) so we use them to store elements that won't change – like the days of the week, the colours of the rainbow etc

Tuples can still be manipulated.
We can concatenate (add) them, we can pick out a single element, (or a slice).
We can also use them as a range of values to be iterated through.
We just can't change them.

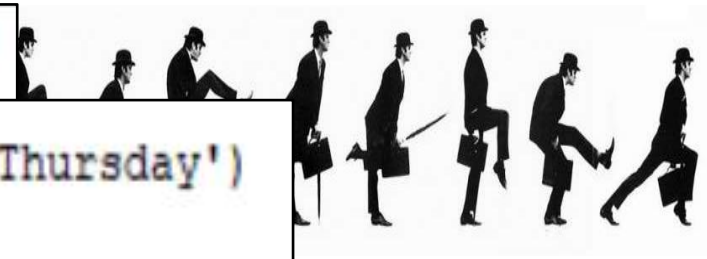
Tuple

```
week_days = ("Monday", "Tuesday", "Wednesday", "Thursday", "Friday")
weekends = ("Saturday", "Sunday")

#we can create a new TUPLE from existing ones
whole_week = week_days + weekends
print()
print("whole_week =", whole_week)



#we can isolate an element
print()
print("The 4th element of whole_week is", whole_week[3])
```

```
whole_week = ('Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday')

The 4th element of whole_week is Thursday
```

Notice: round brackets ()

```
print()
for day in week_days:
    print(day)
```



Tuple

```
week_days = ("Monday", "Tuesday", "Wednesday", "Thursday", "Friday")
weekends = ("Saturday", "Sunday")
```

```
A [1:4] slice of whole_week is ('Tuesday', 'Wednesday', 'Thursday')


Monday
Tuesday
Wednesday
Thursday
Friday
```

```
#we can slice a TUPLE
mid_week = week_days[1:4]
print()
print("A [1:4] slice of whole_week is", mid_week)

#we can iterate through a TUPLE
print()
for day in week_days:
    print(day)
```

```
week_days = ("Monday", "Tuesday", "Wednesday", "Thursday", "Friday")
weekends = ("Saturday", "Sunday")

#we can create a new TUPLE from existing ones
whole_week = week_days + weekends
print()
print("whole_week =", whole_week)



#we can isolate an element
print()
print("The 4th element of whole_week is", whole_week[3])

#we can slice a TUPLE
mid_week = week_days[1:4]
print()
print("A [1:4] slice of whole_week is", mid_week)

#we can iterate through a TUPLE
print()
for day in week_days:
    print(day)
```

```
"C:\Program Files\Python36\python.exe" "F:/IntroToPython 19-20/tuples dictionaries/tuples.py"

whole_week = ('Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday')

The 4th element of whole_week is Thursday

A [1:4] slice of whole_week is ('Tuesday', 'Wednesday', 'Thursday')

Monday
Tuesday
Wednesday
Thursday
Friday

Process finished with exit code 0
```

Dictionary

Lists (and tuples) use numbers as indices e.g. the_pythons is a list and **the_pythons[2] = Gilliam**

So, **the_pythons** connects:

0	to	“Chapman”	
1	to	“Cleese”	
2	to	“Gilliam”	
3	to	“Idle”	etc

Dictionary

A **Dictionary** is also like a List.

However, in a dictionary we can connect any two types.

That can be very useful.

Dictionary

```
# create an empty DICTIONARY called eng_ital
eng_ital = {}

#add the first element
eng_ital["apple"] = "mela"

#add the other elements
eng_ital["orange"] = "arancia"
eng_ital["strawberry"] = "fragola"
eng_ital["lemon"] = "limone"
eng_ital["raspberry"] = "lampone"
```

Dictionaries allow us to associate any two types (the index or **KEY** doesn't have to be a number).
Here we associate a word in English with the corresponding word in Italian (exactly what you'd expect an English-Italian dictionary to do!).

Dictionary

```
# create an empty DICTIONARY called eng_ital
eng_ital = {}

#add the first element
eng_ital["apple"] = "mela"

#add the other elements
eng_ital["orange"] = "arancia"
eng_ital["strawberry"] = "fragola"
eng_ital["lemon"] = "limone"
eng_ital["raspberry"] = "lampone"
```

Dictionaries allow us to associate any two types (the index or **KEY** doesn't have to be a number).
Here we associate a word in English with the corresponding word in Italian (exactly what you'd expect an English-Italian dictionary to do!).

Dictionaries are made up of **KEYS** and **VALUES**.
In this example the English words are the KEYS and the Italian words are the VALUES.

Dictionary

So, **eng_ital** connects:

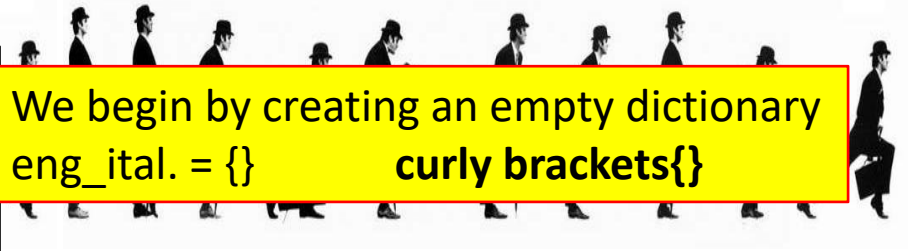
KEY	to	**VALUE**
apple	to	mela
orange	to	arancia
strawberry	to	fragola
lemon	to	limone
raspberry	to	lampone

Dictionary

Here are some things we can do with a Dictionary

```
# create an empty DICTIONARY called eng_ital
eng_ital = {}

#add the first element
eng_ital["apple"] = "mela"

#add the other elements
eng_ital["orange"] = "arancia"
eng_ital["strawberry"] = "fragola"
eng_ital["lemon"] = "limone"
eng_ital["raspberry"] = "lampone"

# display the whole dictionary
print("\nThe WHOLE dictionary")
print(eng_ital)

# identify the KEYS
print("\nJust the KEYS")
print(eng_ital.keys())

# identify the VALUES
print("\nNow just the VALUES")
print(eng_ital.values())

# Check in the KEYS
print("\norange is in the dictionary - ","orange" in eng_ital)
print("peach is in the dictionary - ","peach" in eng_ital)
print("limone is in the dictionary - ","limone" in eng_ital)
```

We begin by creating an empty dictionary eng_ital. = {} **curly brackets{}**



Dictionary

Here are some things we can do with a Dictionary

```
# create an empty DICTIONARY called eng_ital
eng_ital = {}

#add the first element
eng_ital["apple"] = "mela"

#add the other elements
eng_ital["orange"] = "arancia"
eng_ital["strawberry"] = "fragola"
eng_ital["lemon"] = "limone"
eng_ital["raspberry"] = "lampone"

# display the whole dictionary
print("\nThe WHOLE dictionary")
print(eng_ital)

# identify the KEYS
print("\nJust the KEYS")
print(eng_ital.keys())

# identify the VALUES
print("\nNow just the VALUES")
print(eng_ital.values())

# Check in the KEYS
print("\norange is in the dictionary - ","orange" in eng_ital)
print("peach is in the dictionary - ","peach" in eng_ital)
print("limone is in the dictionary - ","limone" in eng_ital)
```

We begin by creating an empty dictionary eng_ital. = {} **curly brackets{}**

Now we populate the dictionary with ordered pairs of elements.
The KEY in square brackets [] is paired with its Italian VALUE.

Dictionary

Here are some things we can do with a Dictionary

```
# create an empty DICTIONARY called eng_ital
eng_ital = {}

#add the first element
eng_ital["apple"] = "mela"

#add the other elements
eng_ital["orange"] = "arancia"
eng_ital["strawberry"] = "fragola"
eng_ital["lemon"] = "limone"
eng_ital["raspberry"] = "lampone"

# display the whole dictionary
print("\nThe WHOLE dictionary")
print(eng_ital)

# identify the KEYS
print("\nJust the KEYS")
print(eng_ital.keys())

# identify the VALUES
print("\nNow just the VALUES")
print(eng_ital.values())

# Check in the KEYS
print("\norange is in the dictionary - ","orange" in eng_ital)
print("peach is in the dictionary - ","peach" in eng_ital)
print("limone is in the dictionary - ","limone" in eng_ital)
```

We begin by creating an empty dictionary eng_ital. = {} **curly brackets{}**

Now we populate the dictionary with ordered pairs of elements.
The KEY in square brackets [] is paired with its Italian VALUE.

We can print the whole dictionary, Keys and Values ….

Dictionary

Here are some things we can do wi
Dictio

```
# create an empty DICTIONARY called eng_ital
eng_ital = {}

#add the first element
eng_ital["apple"] = "mela"

#add the other elements
eng_ital["orange"] = "arancia"
eng_ital["strawberry"] = "fragola"
eng_ital["lemon"] = "limone"
eng_ital["raspberry"] = "lampone"

# display the whole dictionary
print("\nThe WHOLE dictionary")
print(eng_ital)

# identify the KEYS

# identify the VALUES
print("\nNow just the VALUES")
print(eng_ital.values())

# Check in the KEYS
print("\norange is in the dictionary - ","orange" in eng_ital)
print("peach is in the dictionary - ","peach" in eng_ital)
print("limone is in the dictionary - ","limone" in eng_ital)
```

```
The WHOLE dictionary
{'apple': 'mela', 'orange': 'arancia', 'strawberry': 'fragola', 'lemon': 'limone', 'raspberry': 'lampone'}
```

We begin by creating an empty dictionary eng_ital. = {} **curly brackets{}**

Now we populate the dictionary with ordered pairs of elements.
The KEY in square brackets [] is paired with its Italian VALUE.

We can print the whole dictionary, Keys and Values ….

Dictionary

Here are some things we can do with a Dictionary

```
# create an empty DICTIONARY called eng_ital
eng_ital = {}

#add the first element
eng_ital["apple"] = "mela"

#add the other elements
eng_ital["orange"] = "arancia"
eng_ital["strawberry"] = "fragola"
eng_ital["lemon"] = "limone"
eng_ital["raspberry"] = "lampone"

# display the whole dictionary
print("\nThe WHOLE dictionary")
print(eng_ital)

# identify the KEYS
print("\nJust the KEYS")
print(eng_ital.keys())

# identify the VALUES
print("\nNow just the VALUES")
print(eng_ital.values())

# Check in the KEYS
print("\norange is in the dictionary - ","orange" in eng_ital)
print("peach is in the dictionary - ","peach" in eng_ital)
print("limone is in the dictionary - ","limone" in eng_ital)
```

We begin by creating an empty dictionary eng_ital. = {} **curly brackets{}**

Now we populate the dictionary with ordered pairs of elements.
The KEY in square brackets [] is paired with its Italian VALUE.

We can print the whole dictionary, Keys and Values ….

.. or just the Keys ….

Dictionary

Here are some things we can do with a Dictionary

```
# create an empty DICTIONARY called eng_ital
eng_ital = {}

#add the first element
eng_ital["apple"] = "mela"

#add the other elements
eng_ital["orange"] = "arancia"
eng_ital["strawberry"] = "fragola"
eng_ital["lemon"] = "limone"
eng_ital["raspberry"] = "lampone"

# display the whole dictionary
print("\nThe WHOLE dictionary")
print(eng_ital)

# identify the KEYS
print("\nJust the KEYS")
print(eng_ital.keys())
```

```
Just the KEYS
dict_keys(['apple', 'orange', 'strawberry', 'lemon', 'raspberry'])
```

```
# Check in the KEYS
print("\norange is in the dictionary - ","orange" in eng_ital)
print("peach is in the dictionary - ","peach" in eng_ital)
print("limone is in the dictionary - ","limone" in eng_ital)
```

We begin by creating an empty dictionary eng_ital. = {} **curly brackets{}**

Now we populate the dictionary with ordered pairs of elements.
The KEY in square brackets [] is paired with its Italian VALUE.

We can print the whole dictionary, Keys and Values ….

.. or just the Keys ….

Dictionary

Here are some things we can do with a Dictionary

```
# create an empty DICTIONARY called eng_ital
eng_ital = {}

#add the first element
eng_ital["apple"] = "mela"

#add the other elements
eng_ital["orange"] = "arancia"
eng_ital["strawberry"] = "fragola"
eng_ital["lemon"] = "limone"
eng_ital["raspberry"] = "lampone"

# display the whole dictionary
print("\nThe WHOLE dictionary")
print(eng_ital)

# identify the KEYS
print("\nJust the KEYS")
print(eng_ital.keys())

# identify the VALUES
print("\nNow just the VALUES")
print(eng_ital.values())

# Check in the KEYS
print("\norange is in the dictionary - ","orange" in eng_ital)
print("peach is in the dictionary - ","peach" in eng_ital)
print("limone is in the dictionary - ","limone" in eng_ital)
```

We begin by creating an empty dictionary eng_ital. = {} **curly brackets{}**

Now we populate the dictionary with ordered pairs of elements.
The KEY in square brackets [] is paired with its Italian VALUE.

We can print the whole dictionary, Keys and Values ….

.. or just the Keys ….

.. or just the Values.

Dictionary

Here are some things we can do with a Dictionary

```
# create an empty DICTIONARY called eng_ital
eng_ital = {}

#add the first element
eng_ital["apple"] = "mela"

#add the other elements
eng_ital["orange"] = "arancia"
eng_ital["strawberry"] = "fragola"
eng_ital["lemon"] = "limone"
eng_ital["raspberry"] = "lampone"

# display the whole dictionary
print("\nThe WHOLE dictionary")
print(eng_ital)

# identify the KEYS
print("\nJust the KEYS")
print(eng_ital.keys())

# identify the VALUES
print("\nNow just the VALUES")
print(eng_ital.values())
```

```
Now just the VALUES
dict_values(['mela', 'arancia', 'fragola', 'limone', 'lampone'])
```

```
print("limone is in the dictionary - ","limone" in eng_ital)
```

We begin by creating an empty dictionary eng_ital. = {} **curly brackets{}**

Now we populate the dictionary with ordered pairs of elements.
The KEY in square brackets [] is paired with its Italian VALUE.

We can print the whole dictionary, Keys and Values ….

.. or just the Keys ….

.. or just the Values.

Dictionary

Here are some things we can do with a Dictionary

```
# create an empty DICTIONARY called eng_ital
eng_ital = {}

#add the first element
eng_ital["apple"] = "mela"

#add the other elements
eng_ital["orange"] = "arancia"
eng_ital["strawberry"] = "fragola"
eng_ital["lemon"] = "limone"
eng_ital["raspberry"] = "lampone"

# display the whole dictionary
print("\nThe WHOLE dictionary")
print(eng_ital)

# identify the KEYS
print("\nJust the KEYS")
print(eng_ital.keys())

# identify the VALUES
print("\nNow just the VALUES")
print(eng_ital.values())

# Check in the KEYS
print("\norange is in the dictionary - ","orange" in eng_ital)
print("peach is in the dictionary - ","peach" in eng_ital)
print("limone is in the dictionary - ","limone" in eng_ital)
```

We begin by creating an empty dictionary eng_ital. = {} **curly brackets{}**

Now we populate the dictionary with ordered pairs of elements.
The KEY in square brackets [] is paired with its Italian VALUE.

We can print the whole dictionary, Keys and Values ….

.. or just the Keys ….

.. or just the Values.

We can also check if the dictionary contains a key.

Dictionary

Here are some things we can do with a Dictionary

```
# create an empty DICTIONARY called eng_ital
eng_ital = {}

#add the first element
eng_ital["apple"] = "mela"

#add the other elements
eng_ital["orange"] = "arancia"
eng_ital["strawberry"] = "fragola"
eng_ital["lemon"] = "limone"
eng_ital["raspberry"] = "lampone"

# display the whole dictionary
print("\nThe WHOLE dictionary")
print(eng_ital)

# identify the KEYS
print("\nJust the KEYS")
print(eng_ital.keys())
```

```
orange is in the dictionary -  True
peach is in the dictionary -  False
limone is in the dictionary -  False
```

```
# Check in the KEYS
print("\norange is in the dictionary - ","orange" in eng_ital)
print("peach is in the dictionary - ","peach" in eng_ital)
print("limone is in the dictionary - ","limone" in eng_ital)
```

We begin by creating an empty dictionary eng_ital. = {} **curly brackets{}**

Now we populate the dictionary with ordered pairs of elements.
The KEY in square brackets [] is paired with its Italian VALUE.

We can print the whole dictionary, Keys and Values ….

.. or just the Keys ….

.. or just the Values.

We can also check if the dictionary contains a key.

Dictionary

Here is the Italian to English version.

```
ital_eng = {}
ital_eng["mela"] = "apple"
ital_eng["fragola"] = "strawberry"
ital_eng["arancia"] = "orange"
ital_eng["lampone"] = "raspberry"
ital_eng["limone"] = "lemon"
```

Dictionary

Here is the Italian to English version.

```
ital_eng = {}
ital_eng["mela"] = "apple"
ital_eng["fragola"] = "strawberry"
ital_eng["arancia"] = "orange"
ital_eng["lampone"] = "raspberry"
ital_eng["limone"] = "lemon"
```

Or you can do this

```
ital_eng = {"mela":"apple","fragola":"strawberry","arancia":"orange","lampone":"rapsberry","limone":"lemon"}
```

Dictionary

Let’s revisit a previous program – the car example – and see if we can incorporate one of our new data structures.

Car Example

We created a car Class, and used that to create four instances of car.

```
class car:
    # initialisation method
    def __init__(self,registration,colour,make,model):
        self.registration = registration
        self.colour = colour
        self.make = make
        self.model =model


car1 = car("AB01CDE","White","Ford","Focus")
car2 = car("FG02HIJ","Blue","Vauxhall","Corsa")
car3 = car("KL03MNO","Green","Volkswagen","Polo")
car4 = car("PQ04RST","Red","Toyota","Yaris")
```

Dictionary

Remember, in a Dictionary, we can connect any two types.

So

cars = {car1.registration:car1, car2.registration:car2,
car3.registration:car3, car4.registration:car4}

gives us a Dictionary of all the details of each car (e.g. car1) with its registration number (car1.registration) as the unique key.

Dictionary

So, this **dictionary** connects:

Key		Value
car1.registration	to	car1
car2.registration	to	car2
car3.registration	to	car3
car4.registration	to	car4

Dictionary

We can now add some more code to the program to allow us to prompt for and accept a registration number, search for it and display all the attributes of that car (if it finds it!)

Dictionary

```
cars = {car1.registration:car1, car2.registration:car2, car3.registration:car3, car4.registration:car4}

registration = input("Input a Registration Number or Q to quit >  ")

while registration != "Q":
    if registration in cars.keys():
        vehicle = cars[registration]
        print(vehicle.registration, "is a", vehicle.colour, vehicle.make, vehicle.model)
    else:
        print("Vehicle not found")
    registration = input("Input another Registration Number or Q to quit >  ")
print("Goodbye")
```

```
class car:
    # initialisation method
    def __init__(self,registration,colour,make,model):
        self.registration = registration
        self.colour = colour
        self.make = make
        self.model =model


car1 = car("AB01CDE","White","Ford","Focus")
car2 = car("FG02HIJ","Blue","Vauxhall","Corsa")
car3 = car("KL03MNO","Green","Volkswagen","Polo")
car4 = car("PQ04RST","Red","Toyota","Yaris")

cars = {car1.registration:car1, car2.registration:car2, car3.registration:car3, car4.registration:car4}

registration = input("Input a Registration Number or Q to quit >  ")

while registration != "Q":
    if registration in cars.keys():
        vehicle = cars[registration]
        print(vehicle.registration, "is a", vehicle.colour, vehicle.make, vehicle.model)
    else:
        print("Vehicle not found")
    registration = input("Input another Registration Number or Q to quit >  ")
print("Goodbye")
```

For next week

Create a program to store the details of flights (min of 6 flights) e.g.

Flight Number	Airline	Leaving from	Going to
FR5771	Ryanair	Glasgow	Dublin
KL1473	KLM	Amsterdam	Glasgow
EK023	Emirates	Dubai	Edinburgh
EZY6907	EasyJet	Edinburgh	Geneva

The program should be able to search for a flight number and display all the details of that flight, if the flight number is valid.

Questions??