Topic 6 - Functions

A function is a block of code that performs a specific task

Python already has lots of built in functions:

print(), range(), type(), sleep()

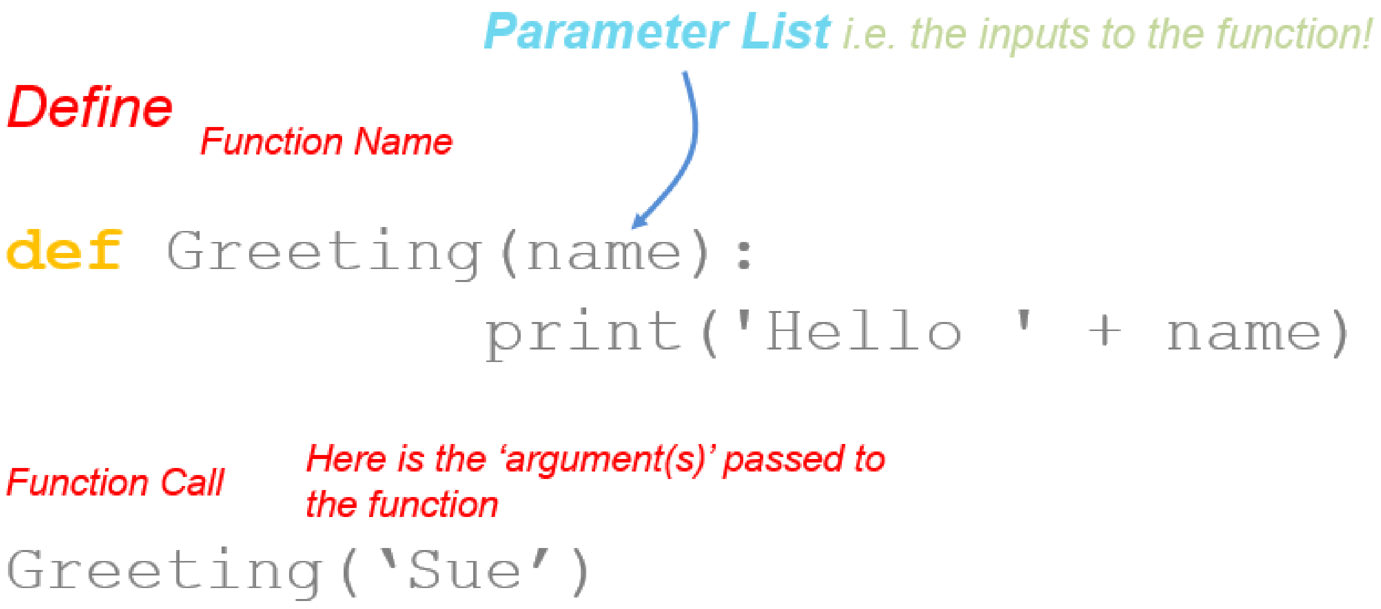
We can declare our own functions to do what we want! These are called **<u>user-defined</u>** functions.

Let's create a new user-defined function that will output a greeting:

In [ ]:

```python
# Example of using functions

# Function Greeting

def Greeting(name):
    print("Hello, " + name)

Greeting("George")
```

Understand the different parts:



Parameter List i.e. the inputs to the function!
Define
Function Name
Function Call
Here is the 'argument(s)' passed to the function

```
def Greeting(name):
            print('Hello ' + name)

Greeting('Sue')
```

In [ ]:

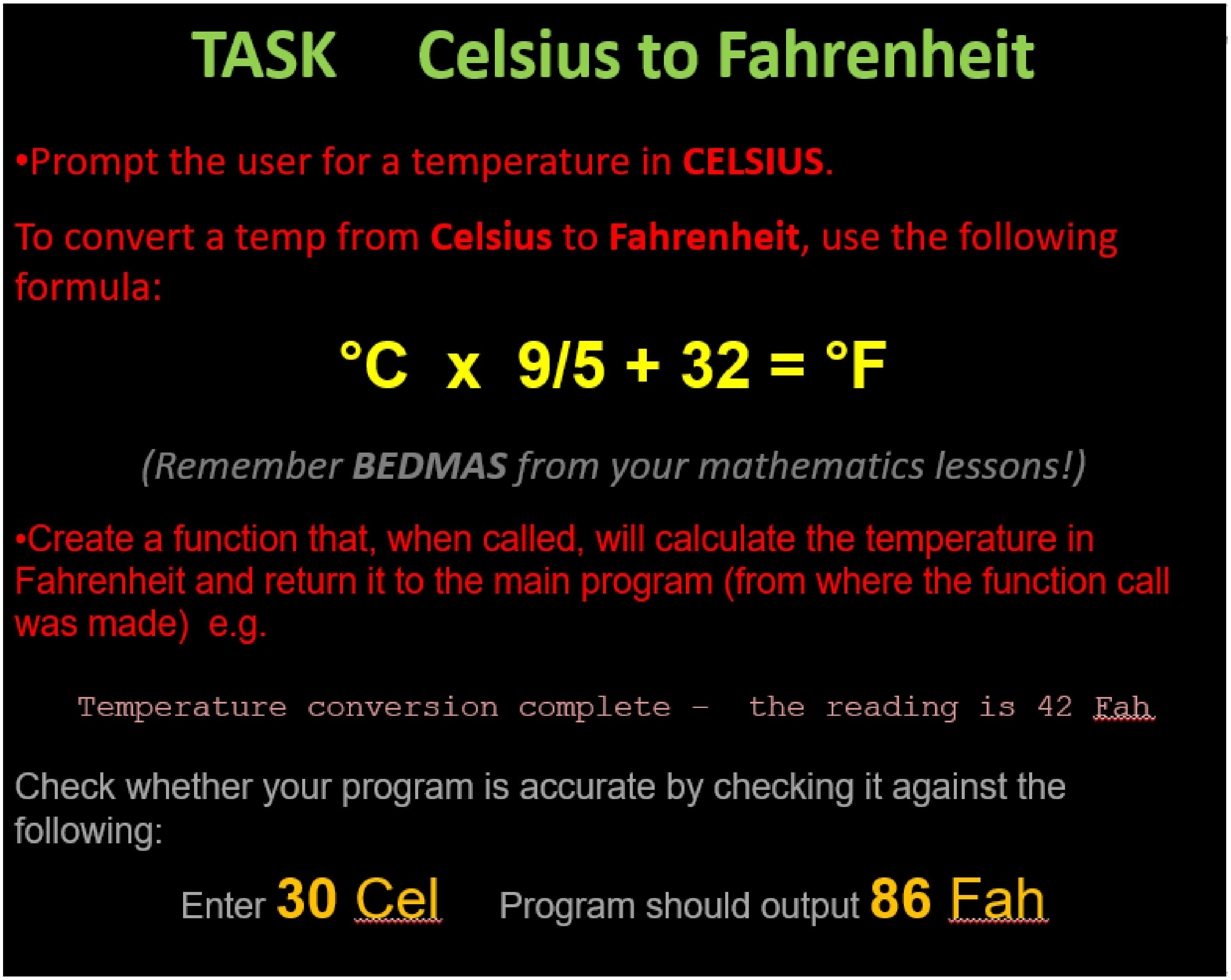
```
# Why won't this compile?

Def Message(name):
    print("How are you today " + name + "?")

Message('Danny')
```

In [1]:

```
# Example of a function

# Pythagoras

def largestNumber(n1,n2):
    if n1 > n2:
        return n1
    else:
        return n2

first_number = int(input("Please enter your first number: "))
second_number = int(input("Please enter your second number: "))

print(largestNumber(first_number, second_number))
```

```
Please enter your first number: 4
Please enter your second number: 3
4
```

Tasks - Celsius to Fahrenheit



TASK Celsius to Fahrenheit

- Prompt the user for a temperature in **CELSIUS**.

To convert a temp from **Celsius** to **Fahrenheit**, use the following formula:

°C x 9/5 + 32 = °F

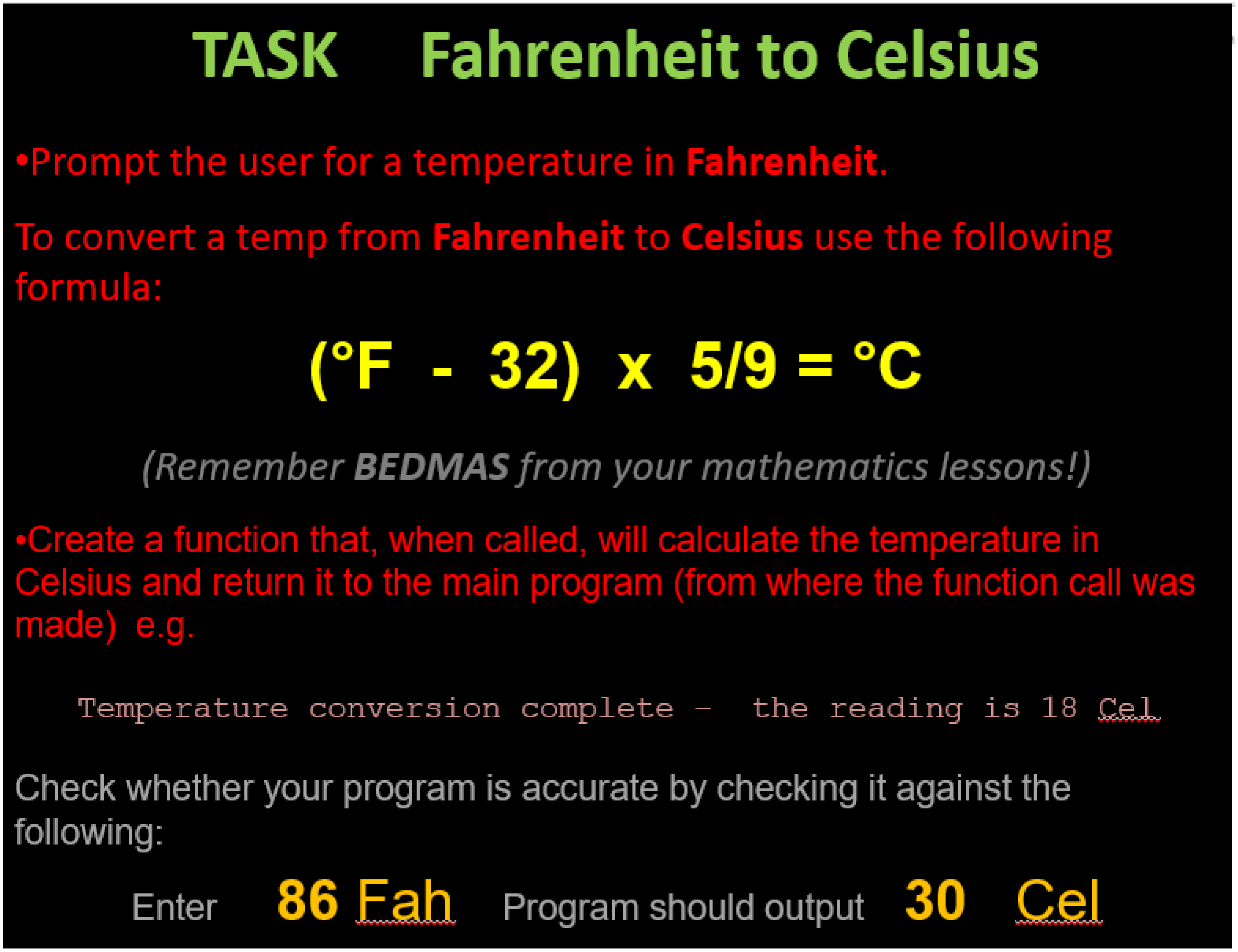
*(Remember **BEDMAS** from your mathematics lessons!)*

- Create a function that, when called, will calculate the temperature in Fahrenheit and return it to the main program (from where the function call was made) e.g.

```
Temperature conversion complete -  the reading is 42 Fah
```

Check whether your program is accurate by checking it against the following:

Enter **30** Cel Program should output **86** Fah



TASK Fahrenheit to Celsius

•Prompt the user for a temperature in **Fahrenheit**.

To convert a temp from **Fahrenheit** to **Celsius** use the following formula:

(°F - 32) x 5/9 = °C

*(Remember **BEDMAS** from your mathematics lessons!)*

•Create a function that, when called, will calculate the temperature in Celsius and return it to the main program (from where the function call was made) e.g.

```
Temperature conversion complete –  the reading is 18 Cel
```

Check whether your program is accurate by checking it against the following:

Enter 86 Fah Program should output 30 Cel

Topic 7 - Lists

Data can be messy, and we need to organize it, especially as your programs get bigger and more complex. We've already touched on **lists**, so let's dive a little deeper.

```
In [ ]: friends = ["Tom", "Sue", "Danny", "Joe", "Mary"]
        print(friends)
```

```
In [ ]: friends = ["Tom", "Sue", "Danny", "Joe", "Mary"]
                #    0      1       2        3      4
        friends[2]
```

```
In [ ]: friends = ["Tom", "Sue", "Danny", "Joe", "Mary"]
        friends[2:4]
```

In [ ]:

```
friends = ["Tom", "Sue", "Danny", "Joe", "Mary"]
friends[-4]
```

append adds an element onto the end of the list

In [ ]:

```
friends = ["Tom", "Sue", "Danny", "Joe", "Mary"]
friends.append("Barry")
print(friends)
```

insert adds an element anywhere in the list that you specify!

In [ ]:

```
friends = ["Tom", "Sue", "Danny", "Joe", "Mary", "Barry"]
friends.insert(2,"Jo")
print(friends)
```

Inserting all the letters of a phrase into a list:

In [ ]:

```
phrase = "joker"

myList = [] # an empty list

for x in range(len(phrase)):
    myList.insert(x, phrase[x])

print(myList)
```

Task 1 - Creating a list

1. Create a list called "friends"
2. At design time (i.e. when you are coding) add 5 of your friends/neighbours to the list
3. Output the contents of the list

Task 2 - Adding to your list

1. Amend your program so that the program prompts the end user for a name during run time
2. The name entered at runtime should be added to the end of the list
3. Decide which **list.method** you will use to achieve this

Task 3 - Refining your list

1. Amend your program so that the program continually prompts the end user for a name
2. The end user will be asked: "Do you want to add a new friend? Type Y to continue or N to quit."
3. If the end user types 'Y' then it will prompt them for a new name to be added to the list
4. If the end user types 'N' then it will say "You have no more friends...goodbye!"

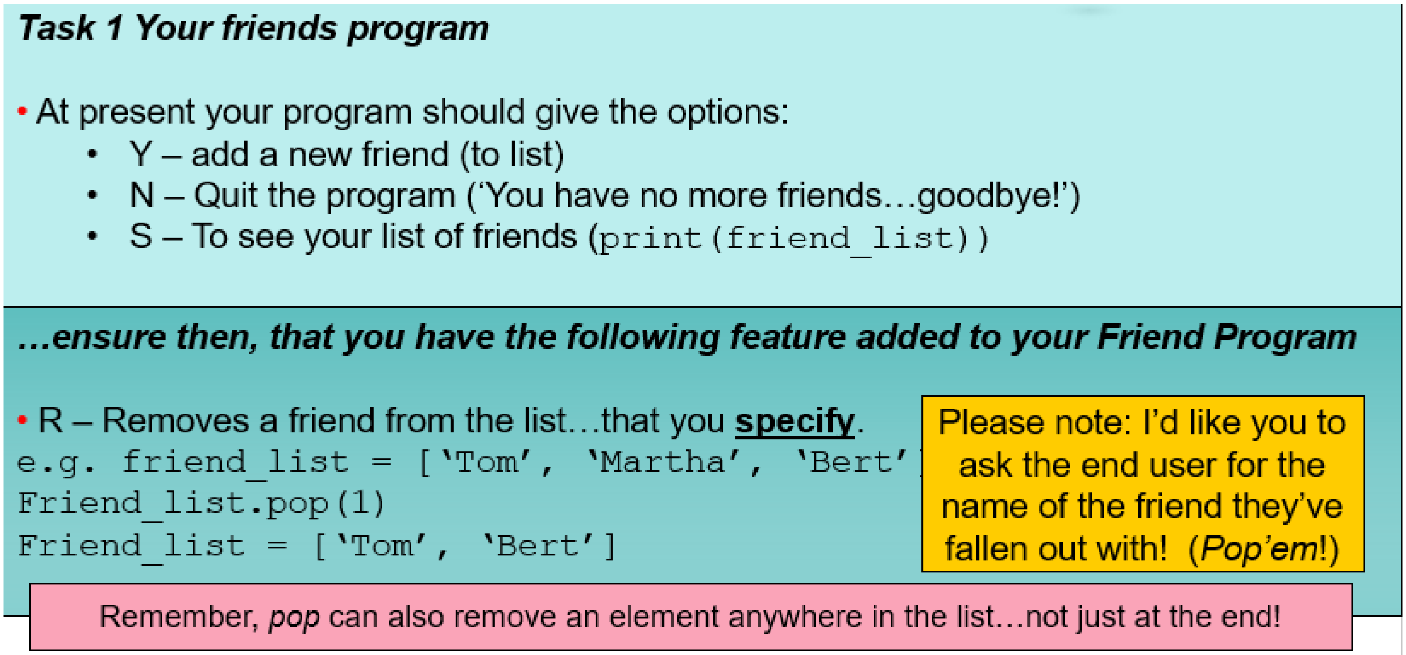
Task 4 - A Third Option (see the list)

1. Amend your program so that there is a third option. I.E. Y to Add to list, N to finish adding, and S to see
   - (i) The **entire friends list** and
   - (ii) **the number of friends in the list**

The in Operator

This will allow you to test if an item is in a list

```
In [ ]: my_list = ['Jo', 'Fred', 'Mary']

        'Mary' in my_list
```



Task 1 Your friends program

- At present your program should give the options:
  - Y – add a new friend (to list)
  - N – Quit the program (‘You have no more friends…goodbye!’)
  - S – To see your list of friends (`print(friend_list)`)

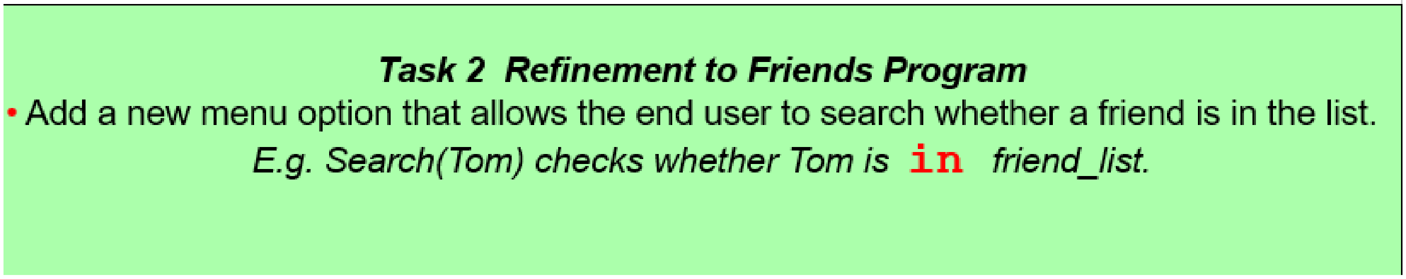
…ensure then, that you have the following feature added to your Friend Program

- R – Removes a friend from the list…that you **specify**.

```
e.g. friend_list = [‘Tom’, ‘Martha’, ‘Bert’]
Friend_list.pop(1)
Friend_list = [‘Tom’, ‘Bert’]
```

Please note: I’d like you to ask the end user for the name of the friend they’ve fallen out with! (*Pop’em*!)

Remember, *pop* can also remove an element anywhere in the list…not just at the end!



Task 2 Refinement to Friends Program

- Add a new menu option that allows the end user to search whether a friend is in the list.

E.g. Search(Tom) checks whether Tom is `in` *friend_list.*

Topic 8 - Dictionaries

Lists or Dictionaries?

If you want a data structure to hold numerous items in order, **USE A LIST**

BUT sometimes, the **order of entry** is not important. In fact, at times you might face a situation in which you want a **mapping** from **keys** to **values**.

A classic example is when you are using a telephone book to look up a name (key) to find an associated phone number (value)

Examples of Keys to Value Mapping

1. 'Hola' : 'Hello'

1. - 'AB' : 'Alberta'
   - 'BC' : 'British Columbia'
   - 'SK' : 'Saskatchewan'
   - 'QC' : 'Quebec'

1. - 'A*' = 10
   - 'A' = 8
   - 'B' = 6
   - 'C' = 5

We find that the **dictionary** data structure is very useful.

So to understand dictionaries, you must accept the **KEY:VALUE** concept.

In Python, we can set up our own dictionary like this:

In [ ]:
```
domain_dictionary = {
    'UK' : 'United Kingdom',
    'IE' : 'Ireland',
    'ES' : 'Spain'
}

print(domain_dictionary)
```

Remember that the entries in the dictionary are **unordered**, as you can see with the above output

To **edit & insert** an entry into the dictionary

dictionary_name[KEY] = new entry

In [ ]:

```
pupils = {
    'Sid' : 18,
    'Tim' : 13,
    'Mary' : 16,
    'Jo' : 15
}

print(pupils)

pupils['Tim'] = 17
print(pupils)
```

In [ ]:

```
pupils = {
    'Sid' : 18,
    'Tim' : 13,
    'Mary' : 16,
    'Jo' : 15
}

pupils['Gandalf'] = 2019

print(pupils)
```

To **delete** an entry in the dictionary, you use the delete function

del dictionary_name[KEY]

In [ ]:

```
pupils = {
    'Sid' : 18,
    'Tim' : 13,
    'Mary' : 16,
    'Jo' : 15,
    'Gandalf' : 2019
}

del pupils['Tim']

pupils
```

Adding new entries to the dictionary using FOR loop

In [ ]:

```
pupils = {}

for x in range(3):
    new_key = input("Enter a new key: ")
    new_age = input("Enter a new age: ")
    pupils[new_key] = new_age

print(pupils)
```

If you use the **dictionary.keys()** method, it will give you a list of the keys as a ***list***

Note the [] in the list

In [ ]:

```
pupils = {
    'Sid' : 18,
    'Tim' : 13,
    'Mary' : 16,
    'Jo' : 15,
    'Gandalf' : 2019
}

pupils.keys()
```

Perhaps you want to **SORT** a dictionary at some point. Well, you will have to use the *sort method*

So, we are going to sort our list by looking at the **VALUES**

In [ ]:

```
pupils = {
    'Sid' : 18,
    'Tim' : 13,
    'Mary' : 16,
    'Jo' : 15,
    'Gandalf' : 2019
}

print(pupils)

sorted(pupils.values())
```

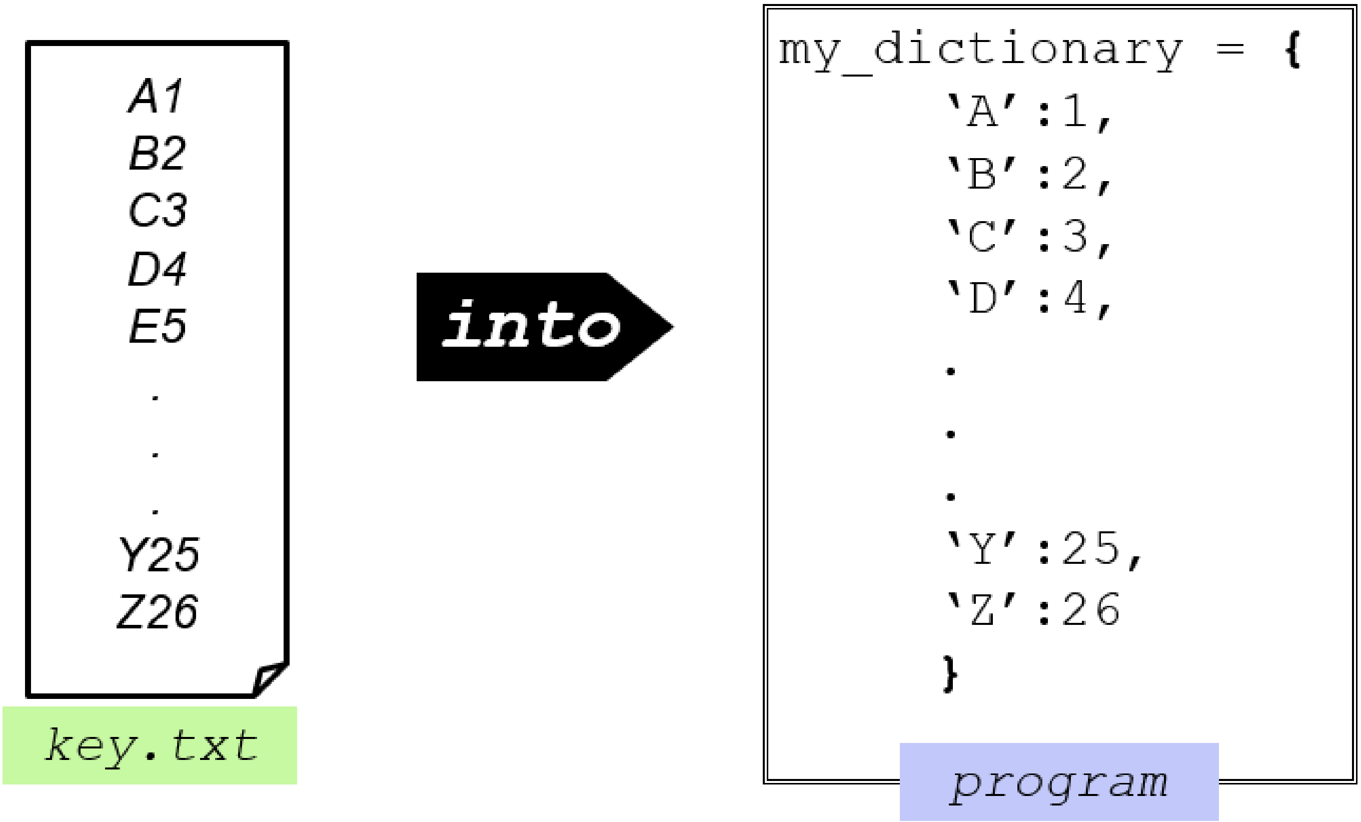
We can also sort by **KEYS**

In [ ]:

```
pupils = {
    'Sid' : 18,
    'Tim' : 13,
    'Mary' : 16,
    'Jo' : 15,
    'Gandalf' : 2019
}

print(pupils)

sorted(pupils.keys())
```

Let's say we have an external file, and we want to **import** data from the external file into a **dictionary**.

How is this achieved?

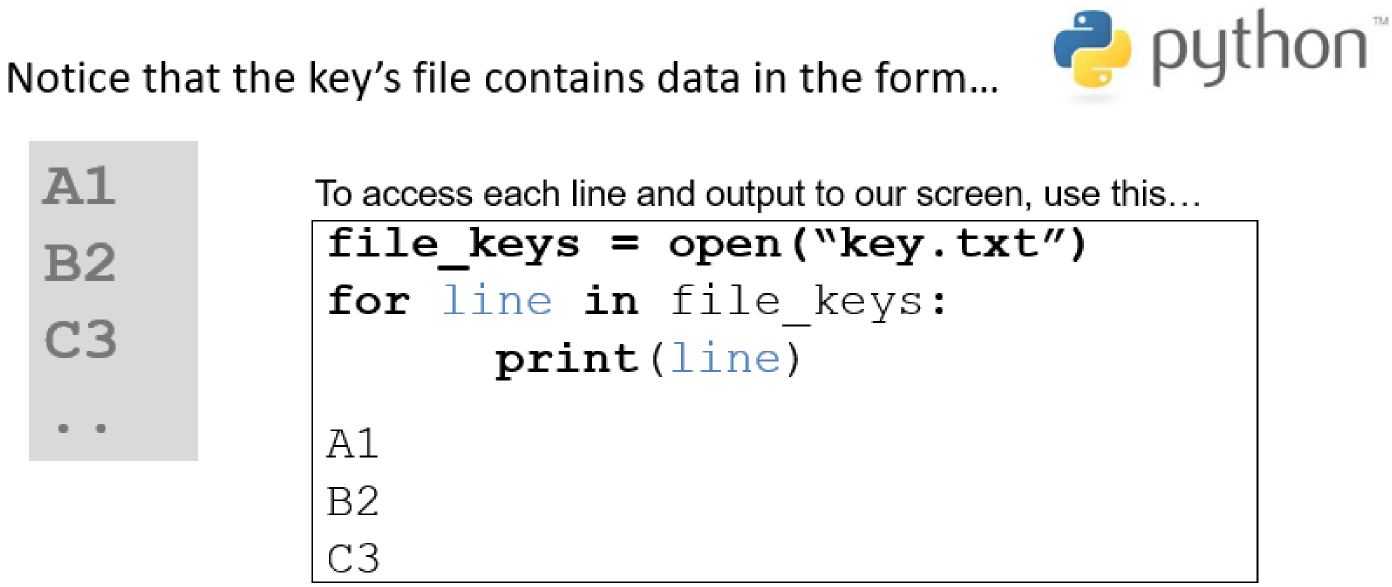


```
A1
B2
C3
D4
E5
.
.
.
Y25
Z26
```

key.txt

```
my_dictionary = {
        'A':1,
        'B':2,
        'C':3,
        'D':4,
        .
        .
        .
        'Y':25,
        'Z':26
        }
```

program

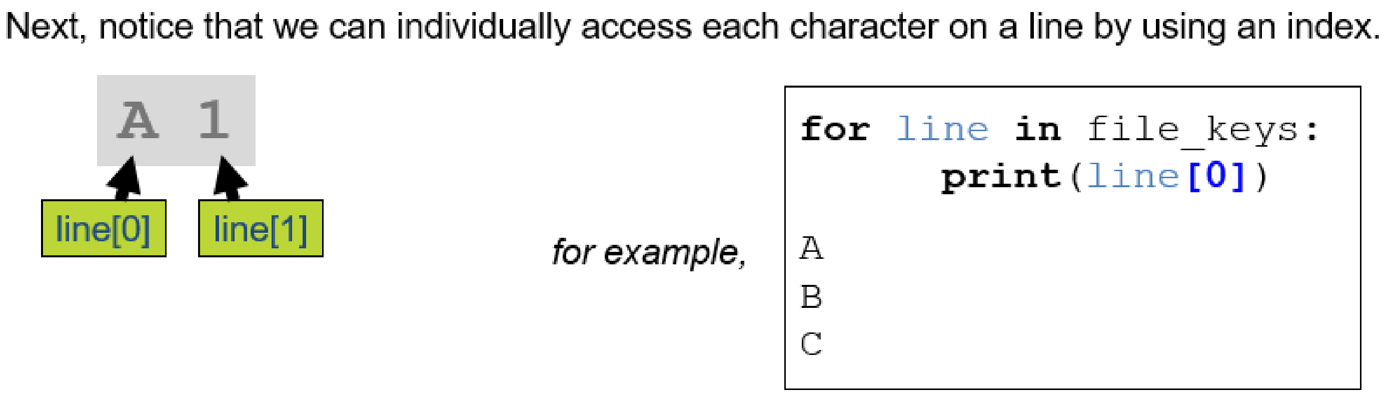


Notice that the key's file contains data in the form…

```
A1
B2
C3
..
```

To access each line and output to our screen, use this…

```
file_keys = open("key.txt")
for line in file_keys:
        print(line)

A1
B2
C3
```



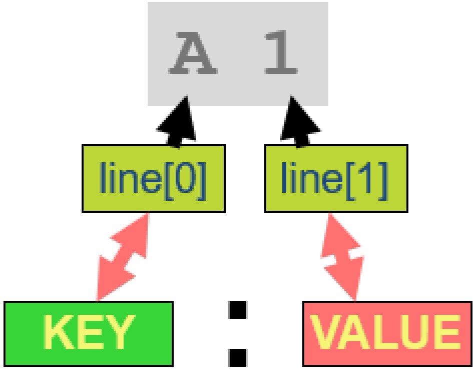
Next, notice that we can individually access each character on a line by using an index.

```
A 1
```

for example,

```
for line in file_keys:
      print(line[0])

A
B
C
```

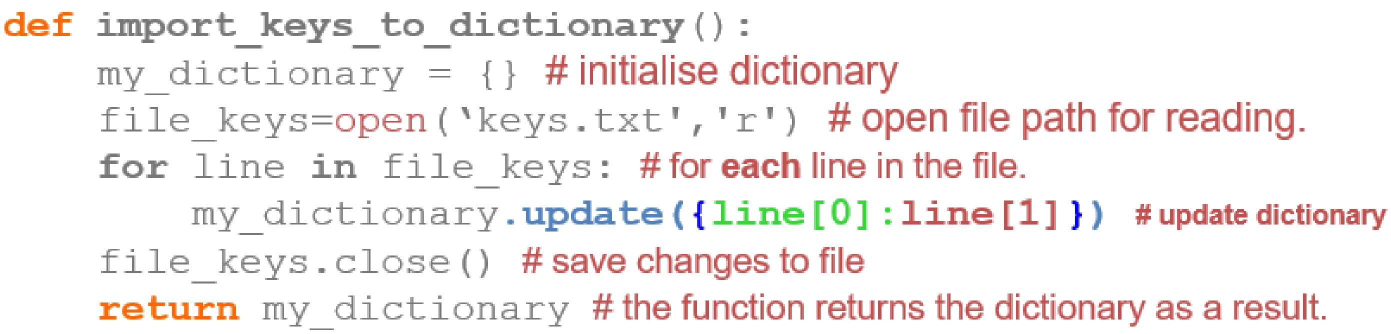
We can take advantage of this and split these two items into **KEY**:**VALUE** format for our dictionary.



Now that our data is split into the **KEY:VALUE** form, we can use a method called *update()*, which will update our dictionary with the new **KEY** and **VALUE** entry:

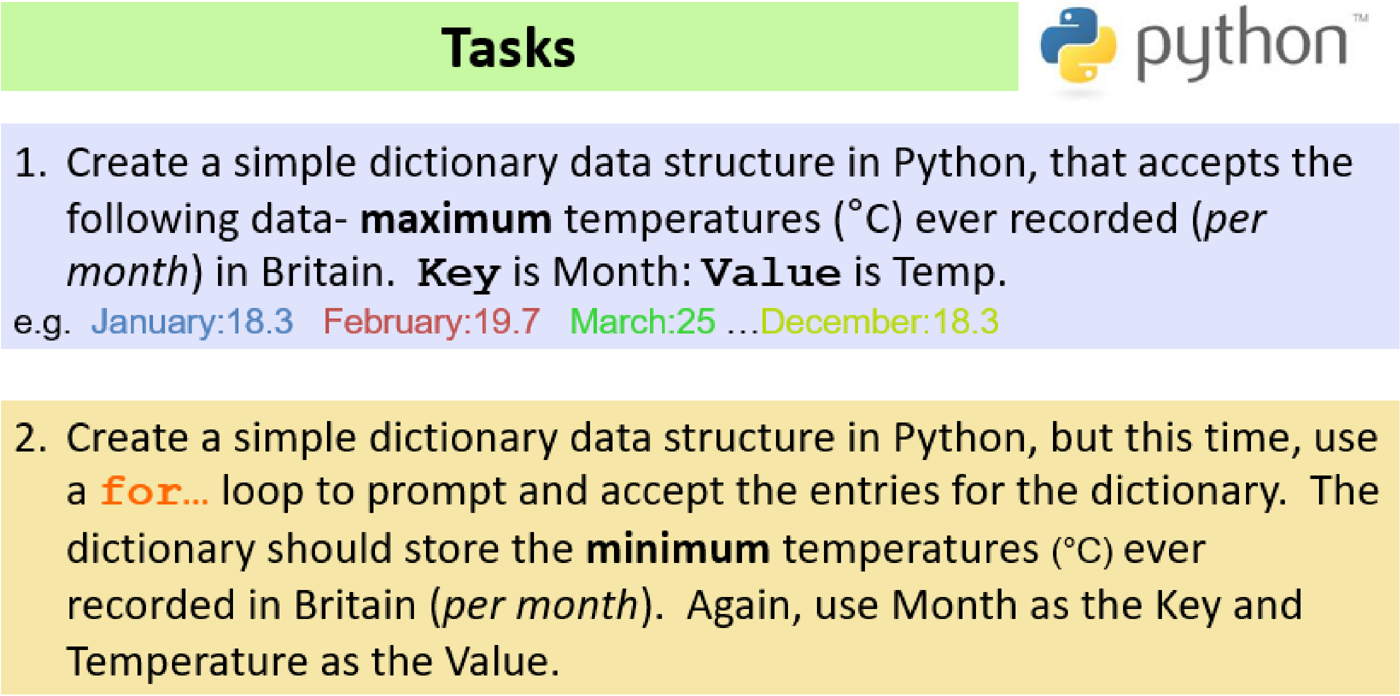
my_dictionary.update({line[0]:line[1]})

Hence, our solution will look like this:



```
def import_keys_to_dictionary():
    my_dictionary = {} # initialise dictionary
    file_keys=open(‘keys.txt','r') # open file path for reading.
    for line in file_keys: # for each line in the file.
        my_dictionary.update({line[0]:line[1]}) # update dictionary
    file_keys.close() # save changes to file
    return my_dictionary # the function returns the dictionary as a result.
```

Task - Temperature Dictionary



Tasks

1. Create a simple dictionary data structure in Python, that accepts the following data- **maximum** temperatures (°C) ever recorded (*per month*) in Britain. **`Key`** is Month: **`Value`** is Temp.

e.g. January:18.3 February:19.7 March:25 …December:18.3

2. Create a simple dictionary data structure in Python, but this time, use a `for`… loop to prompt and accept the entries for the dictionary. The dictionary should store the **minimum** temperatures (°C) ever recorded in Britain (*per month*). Again, use Month as the Key and Temperature as the Value.