# Lesson 2

**REVIEW PREVIOUS LESSON SLIDES!**

Can you create me a variable, to greet by saying “hello world”?

The variable **reference =** *variable\_name.*

The variable **value** = “hello world”.

What datatype is this variable?

* As explained, an animal is classified into a certain species depending on it’s characteristics. Its’ species thus determines it’s functional capabilities.
* Dogs bark(), cats meow(), Birds fly etc.
* By passing through characters enclosed within quotes, you confirmed that you want this object to be of **type** String, which also determines it’s functional capabilities.

Message.upper() = upper case

* Print(message.split()) -🡪 splits the string into individual elements of a list.
* The **split** function essentially splits each word into an independent element of a list.
* This smoothly brings us to our first data structure, lists.

## Lists

* When you progress in programming for larger scale applications, especially the applications you will soon build later, you will have to store information in different ways. This may be a list of players in a leader-board scoresheet in our game.
* Lists are denoted by [] brackets.
* database = [‘Amy, ‘James, ‘Liam]
* Print(database) >>> [‘Amy, ‘James, ‘Liam]

### Access and Indexing

* Baring, in mind that lists are a collection of elements, there are facilities to access specific elements.
* As you can see, we tell the interpreter, that we want the element which is found at position 0. Why 0? Because the computer starts counting from 0, not 1.
* Print(database[0]) >>> ‘Amy
* Print(database[1:]) >>> ‘James, ‘Liam (all elements from position 1, to the end)
* Print(database[-1]) >>> ‘Liam
* Follow me on what I am about to say now. The actual variable “database” is a **LIST** object, however each **element/entry** of ‘database’ is of type **String**. Remember, anything enclosed within quotes is of type string. This means when we access a particular element of a list, we can also exploit the functions/attributes of whatever type the list element is classified as. **(depict visually)**
* Print(database[0].upper()) >>> ‘AMY
* Print(database[0].upper() + ‘IS THE WINNER !’)
* Other examples could be a collection of integers/floats e.g. scores = [30, 190, 3.67]
* When we access elements within a list we can also modify what we want that element to be, as shown below, what if Amy deleted her account, but Josh created a new account? how would we program this?
* Database[0] = ‘Josh’
* Print(database) >>> [‘josh, ‘james, ‘Liam]

SHOW ALL METHODS WHICH ARE AVAILABLE TO USE WITH OUR LIST OBJECT / VARIABLE

TELL HIM TO KEEP THEM DOCUMENTED!

|  |  |
| --- | --- |
| aList.append(x) | Adds the element x to a list |
| aList.count(x) | Returns the number of items x is in list aList |
| aList.index(x) | Returns first position item x is in list |
| aList.insert(k, x) | Insert item x at index k in list aList |
| aList.pop() | Returns the last item in array aList and removes it from the list |
| aList.remove(x) | Removes the first occurrence of x in aList |
| aList.reverse() | Reverses all the items in aList |
| aList.sort() | Sorts list into alphabetical order |

## If Statement

A question rises ….. Let’s say this was a website, where you register an account. Now when you login, we want to check that the name you give, is stored in our database.

What’s your name ?

Well if DARREN is in the database, we would say “successfully logged in!”. Well just as we say it, we can write a program to make this check!

Translate and show how words have significance in English language too.

If ‘DARREN in database:

Print(“successfully logged in!”)>>> successfully logged in!

But what if Darren doesn’t actually have an account? What would the output say ?

We must check for an exception that we **don’t** have those details stored.

Elif ‘Darren’ not in player\_list:

Print(“login unsuccessful!”)

Syntax -🡪 if *condition :*

*do something*

*-----🡪 if this fails, do something ELSE !*

Many students make the mistake of writing many different checks for different possibilities but commencing with if each time. That is wrong, you **must** write commencing with “elif” to denote that you are checking for an exception of a different possibility to the **INITIAL** expectation.

We can also use our mathematical operators to make checks.

If 3 > 2:

Print(“3 is bigger than 2”)

Elif 3 < 2:

Print(“2 is bigger than 3”)

If 1 == 1:

Print(“1 is equal to 1”)

Elif 1 != 1:

Print(“1 is not equal to 1”)