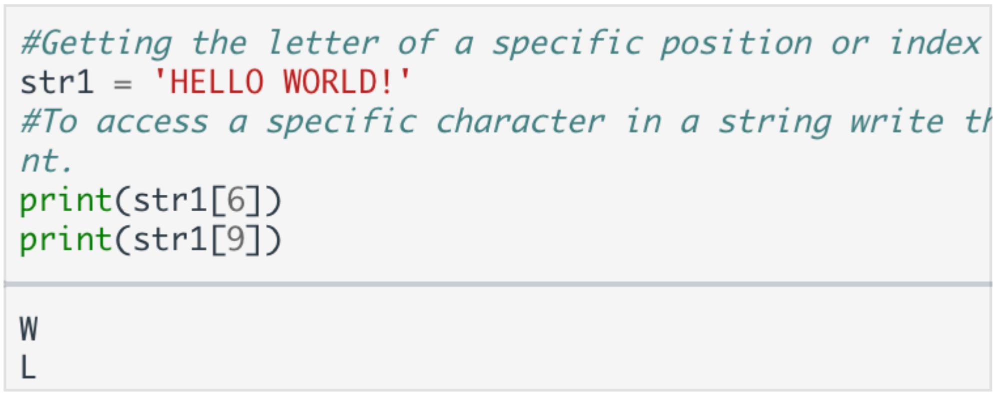
Section A2 String & Lists

String

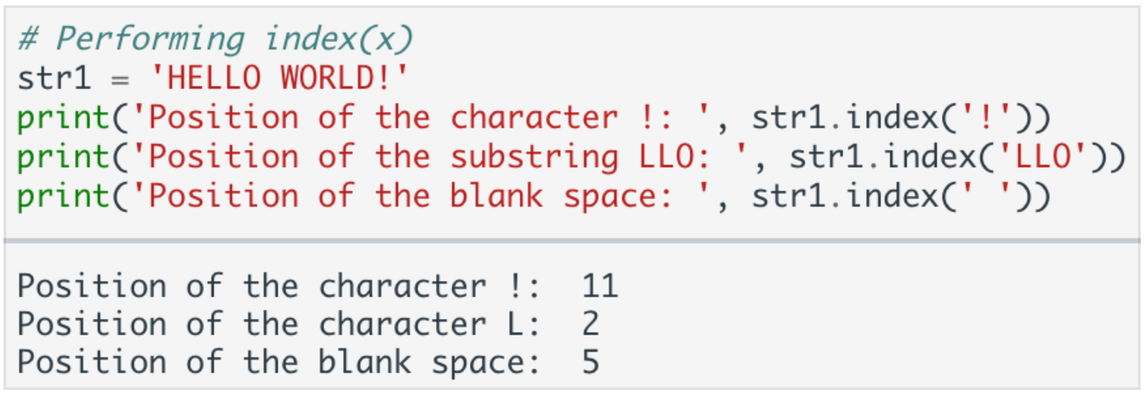
Accessing the characters in a string



```
#Getting the letter of a specific position or index
str1 = 'HELLO WORLD!'
#To access a specific character in a string write th
nt.
print(str1[6])
print(str1[9])
```

```
W
L
```

Finding the first occurrence of a specified value in a string
We can also do the reverse; finding the first index, where a specific character or a substring is located in the string using **index(),** Note that it performs a case-sensitive search.



```
# Performing index(x)
str1 = 'HELLO WORLD!'
print('Position of the character !: ', str1.index('!'))
print('Position of the substring LLO: ', str1.index('LLO'))
print('Position of the blank space: ', str1.index(' '))
```

```
Position of the character !:  11
Position of the character L:  2
Position of the blank space:  5
```

In index(), you can also define from where the search should start and end as follows. If you did not define the start and end, it will search in the entire string as you saw above. **string.index(value, start, end)**
What happens if index() could not find a substring that we specified? It throws **ValueError**
In addition to index(), you can also use **find()** that perform the same operation, The only difference between index() and find() is that find() returns **-1**, if a substring is not found.

Slicing a String
A *slice* is used to pick out part of a string. It usually takes three values:

1. start — starting index where the slicing of the string starts.
2. stop — index until which the slicing takes place. The slicing stops at the length of the string.
3. step — integer value which determines the increment between each index for slicing.

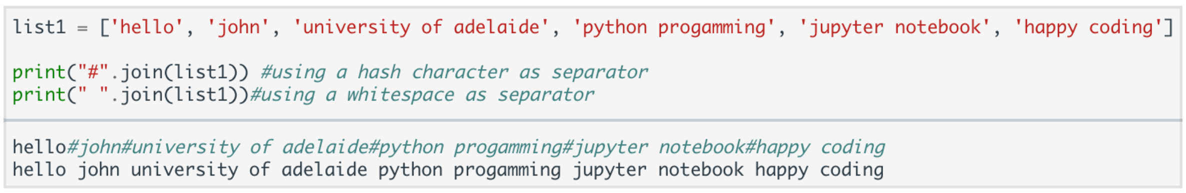
str1[start:stop:step]

You can also use negative values. For the start and stop, it will be the distance of the index from the end of the string, instead of from the beginning. For step, a negative number will cause it to go from the end characters towards the start.

Splitting a String into a List
we use the split() function to split a given string into a list.
str.split(separator, maxsplit)

- separator *(optional)*: You can specify the separator that you want to use when splitting the string. The default separator is any whitespace.
- maxsplit *(optional)*: This defines how many splits you want. The outputted list will contain the specified number of items *plus one*. The default will be -1, indicating all occurrences.

Joining an iterable into one String
you can use **join()** to take all items in an iterable and join them into one string. The iterable could be a list, a dictionary (that you already know), or a tuple, a set



```
list1 = ['hello', 'john', 'university of adelaide', 'python progamming', 'jupyter notebook', 'happy coding']

print("#".join(list1)) #using a hash character as separator
print(" ".join(list1))#using a whitespace as separator
```

```
hello#john#university of adelaide#python progamming#jupyter notebook#happy coding
hello john university of adelaide python progamming jupyter notebook happy coding
```

Lists

Slicing a List
Slicing gets a part of the list. The slice syntax is [start:stop:step] and returns a subsequence of the list as a new list:

- **start** index is optional and when it is omitted, the default value is 0.
- **stop** index is optional and the default value is the length of the list.
- **step** is the space between the values. This value is optional and the default value is 1. A negative step reverses the direction of the iteration.

Insert List Elements
you can add an element to the end of the list, using append(). the **insert(index, element)** function is used to add an element in a specific position indicated by index.

Remove Specified Index
use the pop() function to remove the specified index

```
#Example remove specified index
colors = ['black', 'white', 'yellow', 'red']
colors.pop(1)
print(colors)
```

```
['black', 'yellow', 'red']
```

List Comprehension
The basic syntax of list compresion is:
[expression for item in iterable]
which returns a new list, leaving the old list unchanged.

```
lst1 = []

for i in range(10):
    if i%2 == 0:
        lst1.append(i)
print(lst1)
```

```
[0, 2, 4, 6, 8]
```

The, syntax of a list comprehension using one if statement looks as follows:
[expression for item in iterable if condition == True]

```
even_numbers = [x for x in range(10) if x%2 == 0]
print(even_numbers)
```

```
[0, 2, 4, 6, 8]
```

```
lst1 = ["even" if i%2 == 0 else "odd" for i in range(10)]
print(lst1)
```

```
['even', 'odd', 'even', 'odd', 'even', 'odd', 'even', 'odd', 'even', 'odd']
```

```
matrix = [[1, 9], [2, 0], [9, 6]]

transposed_matrix = [[row[i] for row in matrix] for i in range(2)]
print(transposed_matrix)
```

```
[[1, 2, 9], [9, 0, 6]]
```

In the above program, for i in range(2) is executed before row[i] for row in matrix. Therefore, a value is assigned to i at first, and then item directed by row[i] is appended into the transposed_matrix variable.

Tuples

In Python, tuples are written with round brackets. Tuple elements are ordered, indexed, unchangeable, and allow duplicate values. Tuple elements can be of any data type. It can contain different data types too.

If you want to have a tuple with only one element, you will have to add a comma after the element. Otherwise, Python won't recognise it as a tuple.

```python
details = ("John")
print(type(details))

details = ("John",)
print(type(details))
```

```
<class 'str'>
<class 'tuple'>
```

Sets

In Python, sets are written with curly brackets. They are unordered, unchangeable (but you can remove elements and add new elements), unindexed and do not allow duplicate values.

Once you have created a set, you cannot change its elements, but you can add new elements or remove elements

```python
details = {True, '190cm', 'male', 'Smith', 'John', 30}
#adding new elements
details.add(12)
print(details)
#removing elements
details.remove(30)
print(details)
```

```
{True, '190cm', 'male', 'Smith', 'John', 12, 30}
{True, '190cm', 'male', 'Smith', 'John', 12}
```

let's perform some basic set operations such as **union, intersection, difference, isdisjoint()** and **issubset()**.

```
x = {"John", "Smith", "David", "Danise", "Simon", "Jane"}
y = {"James", "John", "Jane", "Miles"}
print("Union:", x.union(y)) #union() returns a set that contains all elements from both sets (duplicates are excluded)
print("Intersection:", x.intersection(y)) #intersection() returns a set that contains the elements that exist in both sets
print("Difference:", x.difference(y)) #difference() returns a set that contains the elements that only exist in x, and not in y
print("Is Disjoint?", x.isdisjoint(y)) #isdisjoint() returns True if none of the elements are present in both sets, otherwise it retur
ns False
print("Is Subset?", x.issubset(y)) #issubset() returns True if all elements in the set exists in the specified set, otherwise it retun
s False
```

```
Union: {'James', 'Danise', 'David', 'Smith', 'Simon', 'Jane', 'Miles', 'John'}
Intersection: {'Jane', 'John'}
Difference: {'David', 'Smith', 'Simon', 'Danise'}
Is Disjoint? False
Is Subset? False
```

Quick Comparison

Lists	Dictionaries	Tuples	Sets
Allow duplicate elements	No duplicate elements	Allow duplicate elements	No duplicate elements
Changeable	Changeable	Unchangeable	Unchangeable, but you can remove items and add new items
Indexed	Indexed	Indexed	Unindexed
Ordered	Unordered (In Python 3.6 and earlier) Ordered (from Python version 3.7)	Ordered	Unordered
Square bracket [ ]	Curly brackets { }	Round Brackets ( )	Curly brackets { }

**Changeable (also known as mutable) means that we can change, add, and remove elements in a data type after it has been created.*

**Indexed means the first element has index 0 and so on.*

**Ordered means that the elements in a given data type have a defined order, and that order will not change.*