Indexing and Control

Indexing

Indexing rules are followed by all indexable objects, such as str, list and tuple. Python follows zero-based indexing, which means index numbers start from 0. For an indexable object of length n, the valid indices are 0 to n-1.

H	e	l	l	o	!
0	1	2	3	4	5

In the above example, the length of the string is 6, so the valid indices are 0 to 5.

Accessing one element

To access an element using its index, use the following syntax:

```
objectName[index]
```

Where objectName is the name of the object, and index is the index of the element to be accessed. This expression returns the value at that index, provided the index is valid.

Accessing a range of elements

To access a range of elements using indices, use the following syntax:

```
objectName[start:end:step]
```

where:

- start is the starting index (inclusive)
- end is the ending index (exclusive)
- step is the next nth value to be considered after each value

Examples (consider a = 'RACECAR'):

1] a[0:4] -> 'RACE'

2] a[4:] -> 'CAR'

3] a[2:4] -> 'ACE'

4] a[:5] -> 'RACEC'

5] a[::2] -> 'RCCR'

Each of the start, end and step are optional. Just don't forget the colons.

Negative Indexing

Negative indexing is just the opposite of normal indexing: it starts indexing from behind. It follows the syntax:

```
objectName[-n]
```

when the nth element from last is to be accessed. This means that negative indices start from -1 instead of the usual 0.

Selection Statements

Selection statements are used when statements have to be executed only when a condition is met, or to create a branch in the code based on a condition.

1]

```
if condition:
    statements
```

statements are executed if condition evaluates to True, else just skips it.

2]

```
if condition:
    statements1
else:
    statements2
```

statements1 are executed if condition evaluates to True, else statements2 are executed.

3]

```
if condition1:
    statements1
elif condition2:
    statements2
else:
    statements3
```

statements1 are executed if condition1 is True, else statements2 are executed if condition2 is True, else statements3 are executed.

The range() function

The built-in range() function can be used to return an iterable containing a set of values, which can be accessed using an iterative statement.

Syntax: range(start, end, skip)

- start is the starting index (inclusive, optional)
- end is the ending index (exclusive)
- skip used to skip values (optional)

Iterative Statements

Iterative statements are used to execute a set of statements repeatedly, as long as a condition remains True. They are also used to iterate through iterables, like str, list, tuple, range etc.

1]

```
for variableName in iterable:
    statements
```

In each iteration of the loop, a value from the iterable is assigned to variableName, and it can be used inside the loop.

2]

```
while condition:
    statements
```

statements are executed as long as the condition is True. Then it comes out of the loop.

Jump Statements

Used to change the loop execution pattern

a] continue - Skips directly to next iteration

b] break - Immediately come out of the loop

3]

```
while condition:
    statements1
else:
    statements2
```

statements1 are executed as long as the condition is True. statements2 are executed only if the loop was terminated by the condition becoming False.

SilicoFlare