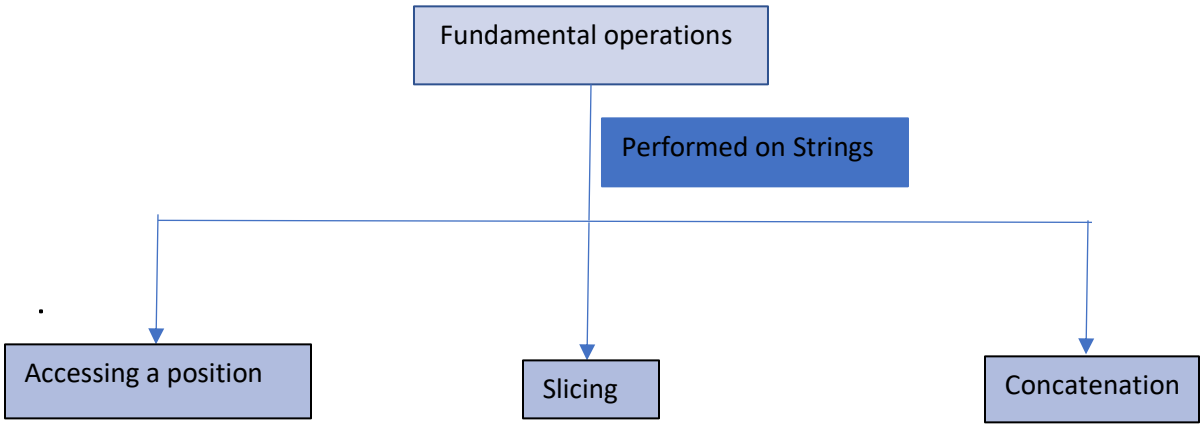
Lab 3

➢ Strings

Strings are one of the fundamental data types in Python, although it is not like this in most programming languages. The purpose of a string is to be a **container** for characters. This makes strings incredibly useful for displaying text to users or manipulating text files.



➢ Accessing a position

```
testString = "This is a string to show operations"
testString[0]
testString[-1]
 testString[4]
```

➢ Slicing

```
myString = "String to slice"
# Slice the whole thing
myString[:]
# Starting index slice
myString[10:]
# Starting and ending index
```

Note that ending index is exclusive

mystring[7:9]

Use only ending point

myString[:6]

Backwards numbers can also be used in

myString[-15:15]

- **For Practice**

```
#strings(grouping of characters)
s="Hello"      #declare " " or ' '
print(s)
print(type(s))

#strings slicing:
s="    HarrY PottEr     "
print(s)
print(s[0:4])
print(s[1:5])
print(s[:4])#to display first 4 characters:
print(s[-4:])#to display last 4 characters:
print(s[2:6:2])
print(s[-1:-5])
print(s[-1:-5:-1])
print(s[-1:-11:-1])
print(s[-2:-6:-2])
print(s[:-3])#starting from till -2
print(s[::2])
print(s[::-1]) #reverse order
print(s[::2])
```

- **Concatenation**

string1 = "I am 1"

string2 = "and I am 2"

integer3 = 3

string1 + string 2

#Notice there isn't an automatic space

string1 + " " + string2

```
# Same as this: f"{string1} {string2}"
# Can combine previously taught operations
string1 + " " + string2 + " and I am" + " " + str(integer3)
# Notice that we want to type cast the integer to a string in order to add them together
string1[:5] + string[-1]
```

- **Operations on Lists (Revision from the last week)**

```
>>> # Adding values to a list
>>> newList = [10, 1, 5, 4, 2323, -223]
>>> newList = newList + ["a"]
>>> print(newList)
[10, 1, 5, 4, 2323, -223, 'a']
>>> newList += ["b"] # This is showing a smaller way of writing
>>> print(newList)
[10, 1, 5, 4, 2323, -223, 'a', 'b']
>>> # We are not using append yet
>>> newList = newList + 1 # This will cause an error .... why?
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate list (not "int") to list
>>> newList = newList + "a" # This will cause an error .... why?
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate list (not "str") to list

>>> # Updating values
>>> aList = [1, 2, 3, 4, 5]
>>> aList[0] = "a"
>>> aList[2] = "b"
>>> print(aList)
['a', 2, 'b', 4, 5]
>>> aList[10] = "lol"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list assignment index out of range
```

```
list
<class 'list'>
#lists works on multiple datatypes(list is hetrogeneous)
#lists always declare in [](square brackets)
l=[12,23,"hello","bye",23.5]
l
[12, 23, 'hello', 'bye', 23.5]
#indexing starts from 0
l[0]
12
l[1]
23
l[2]
'hello'
l[3]
'bye'
l[4]
23.5
l[-1]
23.5
l[-2]
'bye'
l[-3]
'hello'
l[-4]
23
l[-5]
12

#lists slicing
l[1:4]
[23, 'hello', 'bye']
l[1:3]
[23, 'hello']
l[:3] #to display the first 3 characters
[12, 23, 'hello']
l[:4] #to display starting 4 characters.
[12, 23, 'hello', 'bye']
l[1:]
[23, 'hello', 'bye', 23.5]
l[2:3]
['hello']
l[1:4:1]
[23, 'hello', 'bye']
l[1:4:2]
[23, 'bye']
l[1:5:2]
[23, 'bye']
l[2:4:2]
['hello']
l[::-1] #reverse a list.
[23.5, 'bye', 'hello', 23, 12]
l[::2]
[12, 'hello', 23.5]
l[::1]
[12, 23, 'hello', 'bye', 23.5]
l
[12, 23, 'hello', 'bye', 23.5]
```

➢ For Loops

- For loops are what is called a 'bounded' loop. To be bounded means that the the declaration of the loop has a defined beginning and end. A for loop begins with a condition, iterates through the loop, and terminates with an end condition. One common structure of a for loop uses range() and len().

```
animal_list = ["cat", "dog", "hamster", "goat", "turtle"]

for i in range(len(animal_list)):
   print("The animal at index " + str(i), " is " + animal_list[i])
```

- The beginning, iterator and end condition are all contained in range(). Range takes the len() of animal_list: 5. Because range is given only one parameter in this case, the beginning condition is 0, the largest is 4 and the step is 1. The variable i will be assigned to each number produced by range(). From there, I can be used to index into animal_list to print the animal at that index. To print i in the statement, we used the str() function to make sure python treats i as a string rather than a number.

- **For practice**

```
#looping:
#in ->membership operators

for i in range(5):
    print(i) #end='' print the output in same line.

#dry run
i=0     i=1         i=2      i=3     i=4
0<5     1<5         2<5      3<5     4<5
0(p)      1(p)      2(p)      3(p)    4(p)

for i in range(1,5):
    print(i,end='') #end='' print the output in same line.

1->start
5->stop

for i in range(1,5,2):
    print(i,end='hello') #end='' print the output in same line.

1->start
5->stop
2->step value (increment value by 2)

i=1
1<5
1(p)
i=1+2=3

3<5
3(p)
i=3+2=5
5<5
```

- **Range**

Range is a *generator* this is something we will discuss in more detail later in the semester. For now, we can think of it as a function that will create a list for us. The syntax of range follows range([begin], end,[step]). Explain to the students how begin, end and [step] works in the example below.

```
#Show only using end
#note end exclusive
list(range(5))
#Using begin and end
#Not begin inclusive
list(range(5,10))
# Use step to move forward by
# non 1 increments
list(range(0,10,2))
# Good to get only even numbers
```