



SNS COLLEGE OF ENGINEERING
Kurumbapalayam (Po), Coimbatore – 641 107



An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

**DEPARTMENT OF CSE (IoT & CYBER SECURITY INCLUDING BLOCKCHAIN
TECHNOLOGY)**



19IT103 – COMPUTATIONAL THINKING AND PYTHON PROGRAMMING

- ❖ A readable, dynamic, pleasant, flexible, fast and powerful language

Recap

- Function composition is a way of combining functions
- Function composition is achieved through lambda functions
- Lambda functions are called anonymous because they are not declared in the standard manner by using the def keyword
- Recursion is the process calling a function by itself

Agenda

- Strings
- String Immutability

Strings

- A string is a sequence of characters
- A string literal uses quotes `'Hello'` or `"Hello"`
- For strings, `+` means “concatenate”
- When a string contains numbers, it is still a string
- We can convert numbers in a string into a number using `int()`

Strings

Example:

```
>>> str1 = "Hello"  
>>> str2 = 'there'  
>>> bob = str1 + str2  
>>> print bob  
Hellothere  
>>> str3 = '123'  
>>> str3 = str3 + 1  
Traceback (most recent call last):  
File "<stdin>", line 1, in  
<module>TypeError: cannot  
concatenate 'str' and 'int' objects  
>>> x = int(str3) + 1  
>>> print x  
124  
>>>
```

Strings

Reading and Converting

- We prefer to read data in using **strings** and then parse and convert the data as we need
- This gives us more control over error situations and/or bad user input
- Raw input numbers must be **converted** from strings

Strings

Example:

```
>>> name = raw_input('Enter:')
Enter:Chuck
>>> print name
Chuck
>>> apple = raw_input('Enter:')
Enter:100
>>> x = apple - 10
Traceback (most recent call last):
  File "<stdin>", line 1, in
<module>TypeError: unsupported
operand type(s) for -: 'str' and 'int'
>>> x = int(apple) - 10
>>> print x
90
```

Strings

Looking Inside Strings:

- We can get at any single character in a string using an index specified in **square brackets**
- The index value must be an integer and starts at zero
- The index value can be an expression that is computed

```
b a n a n a
0 1 2 3 4 5
>>> fruit = 'banana'
>>> letter = fruit[1]
>>> print letter
a
>>> n = 3
>>> w = fruit[n - 1]
>>> print w
n
```


Strings

A Character Too Far

- You will get a **python error** if you attempt to index beyond the end of a string.
- So be careful when constructing index values and slices

```
>>> zot = 'abc'
>>> print zot[5]
Traceback (most recent call last):
File "<stdin>", line 1, in
<module>IndexError: string index
out of range
>>>
```

Strings

Strings Have Length

- There is a built-in function `len` that gives us the length of a string

```
b a n a n a
0 1 2 3 4 5

>>> fruit = 'banana'
>>> print len(fruit)
6
```

Strings

Len Function

```
>>> fruit = 'banana'  
>>> x = len(fruit)  
>>> print x  
6
```

'banana'
(a string)



len()
function



6
(a number)

Strings

Looping Through Strings:

- Using a **while** statement and an **iteration variable**, and the **len** function, we can construct a loop to look at each of the letters in a string individually

```
fruit = 'banana'           0 b
index = 0                  1 a
while index < len(fruit) : 2 n
    letter = fruit[index]   3 a
    print index, letter     4 n
    index = index + 1       5 a
```

Strings

Looping Through Strings:

- A definite loop using a **for** statement is much more elegant
- The **iteration variable** is completely taken care of by the **for** loop

```
fruit = 'banana'  
for letter in fruit :  
    print letter
```

b
a
n
a
n
a

Strings

Looping Through Strings:

- A definite loop using a **for** statement is much more **elegant**
- The **iteration variable** is completely taken care of by the **for** loop

```
fruit = 'banana'
for letter in fruit :
    print letter

index = 0
while index < len(fruit) :
    letter = fruit[index]
    print letter
    index = index + 1
```

b
a
n
a
n
a

Strings

Looping Through Strings:

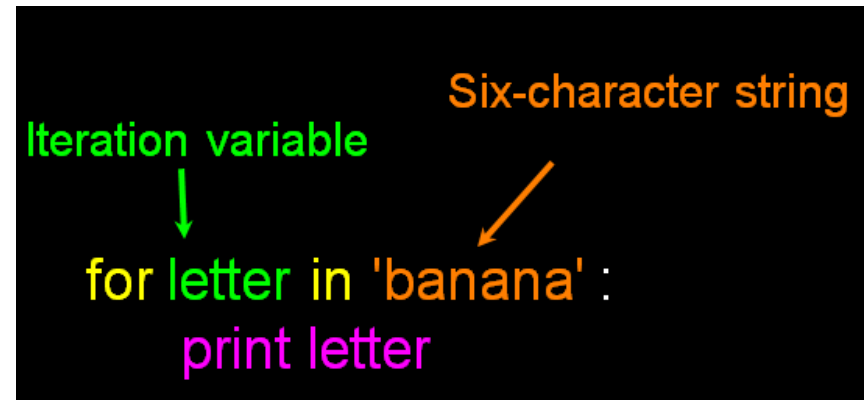
- This is a simple loop that loops through each letter in a string and counts the number of times the loop encounters the 'a' character

```
word = 'banana'  
count = 0  
for letter in word :  
    if letter == 'a' :  
        count = count + 1  
print count
```

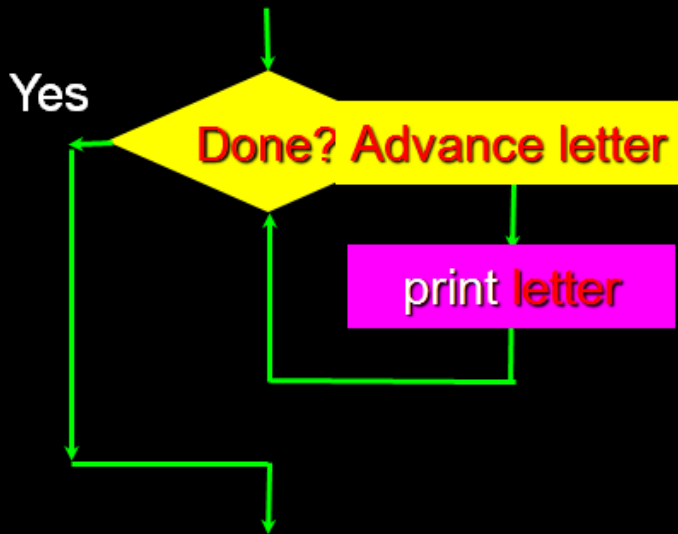
Strings

Looking deeper into **in**

- The **iteration variable** “iterates” through the **sequence** (ordered set)
- The **block (body)** of code is executed once for each value **in** the **sequence**
- The **iteration variable** moves through all of the values **in** the **sequence**



Strings



b	a	n	a	n	a
---	---	---	---	---	---

```
for letter in 'banana':  
    print letter
```

The **iteration variable** “iterates” through the **string** and the **block (body)** of code is executed once for each value **in** the **sequence**

String Immutability

- In python, the string data types are immutable. i.e., a string value cannot be updated.
- We can verify this by trying to update a part of the string which will led us to an error.

```
# Can not reassign
demo_str= "See"
print(type(demo_str))
demo_str[2] = "a"
```

...

<class 'str'>

Traceback (most recent call last):

File "C:/Python34/demotk.py", line 4, in <module>

demo_str[2] = "a"

TypeError: 'str' object does not support item assignment

>>>

Summary

- Strings Read/Convert
- Indexing strings using []
- Looping through strings with **for** and **while**
- Concatenating strings with **+**
- Strings are immutable

A yellow speech bubble with a pointed tail pointing towards the bottom right, set against a solid blue background. The words "THANK YOU" are cut out of the bubble in a bold, sans-serif font, revealing the blue background underneath. The bubble has rounded corners and a slight shadow, giving it a 3D appearance.

THANK YOU