Strings

Chapter 6

Python for Everybody

www.py4e.com

String Data Type

- 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()

```
>>> 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
>>>
```

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
- Input numbers must be converted from strings

```
>>> name = input('Enter:')
Enter:Chuck
>>> print(name)
Chuck
>>> apple = 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
```

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
>>> x = 3
>>> w = fruit[x - 1]
>>> print(w)
n
```

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 Have Length

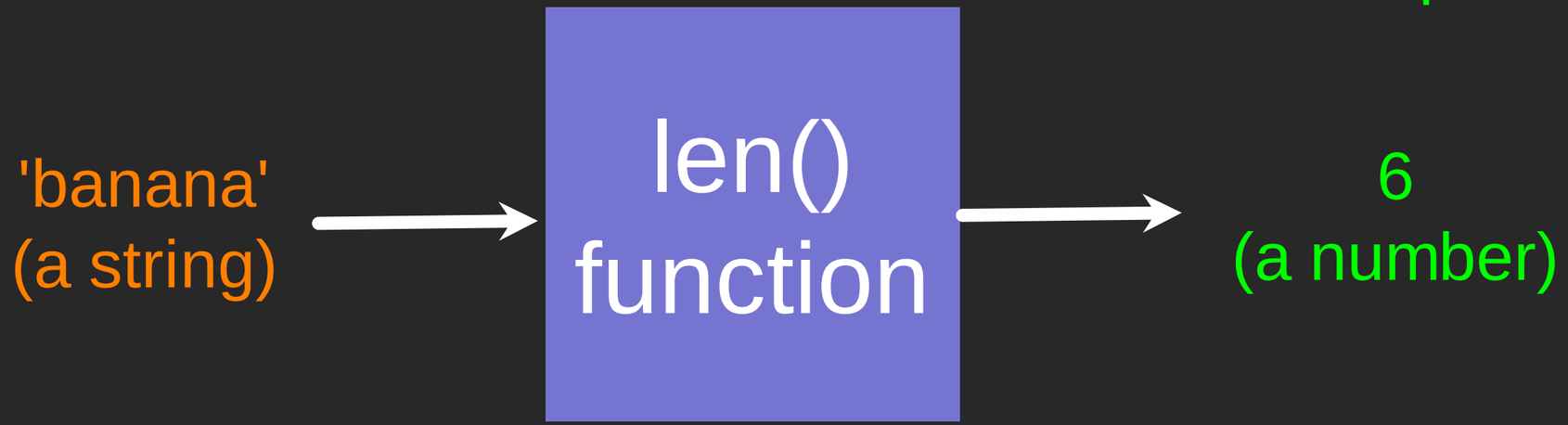
The built-in function len gives us the length of a string


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

len Function (1 of 2)

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

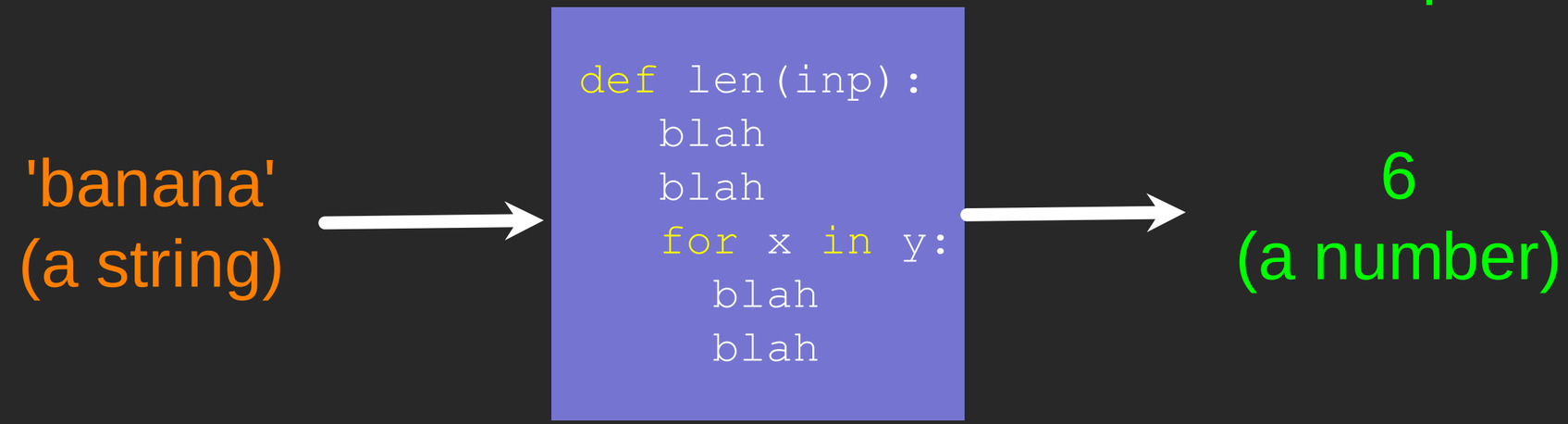
A function is some stored code that we use. A function takes some input and produces an output.



len Function (2 of 2)

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

A function is some stored code that we use. A function takes some input and produces an output.



Looping Through Strings (1 of 3)

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

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

Looping Through Strings (2 of 3)

- 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
```

Looping Through Strings (3 of 3)

- 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
```

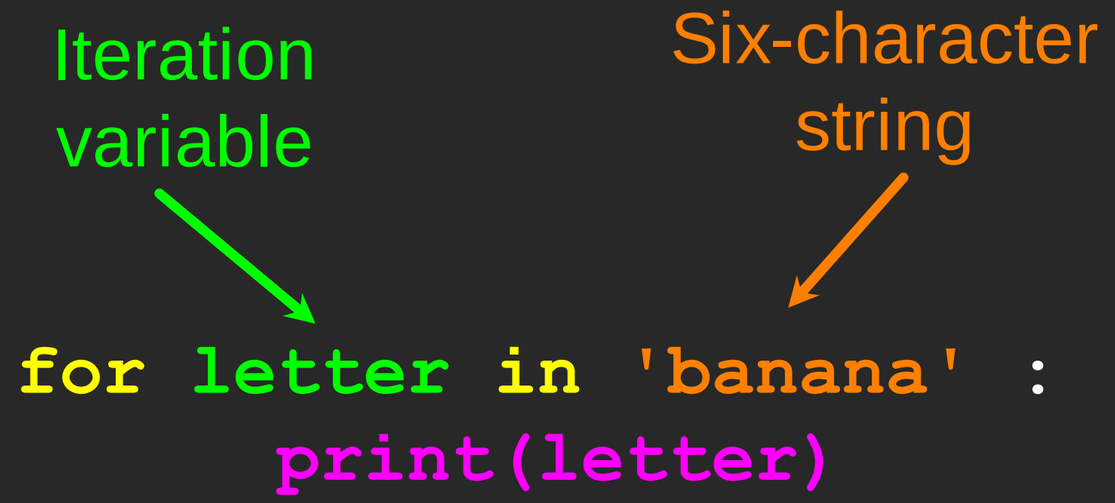
Looping and Counting

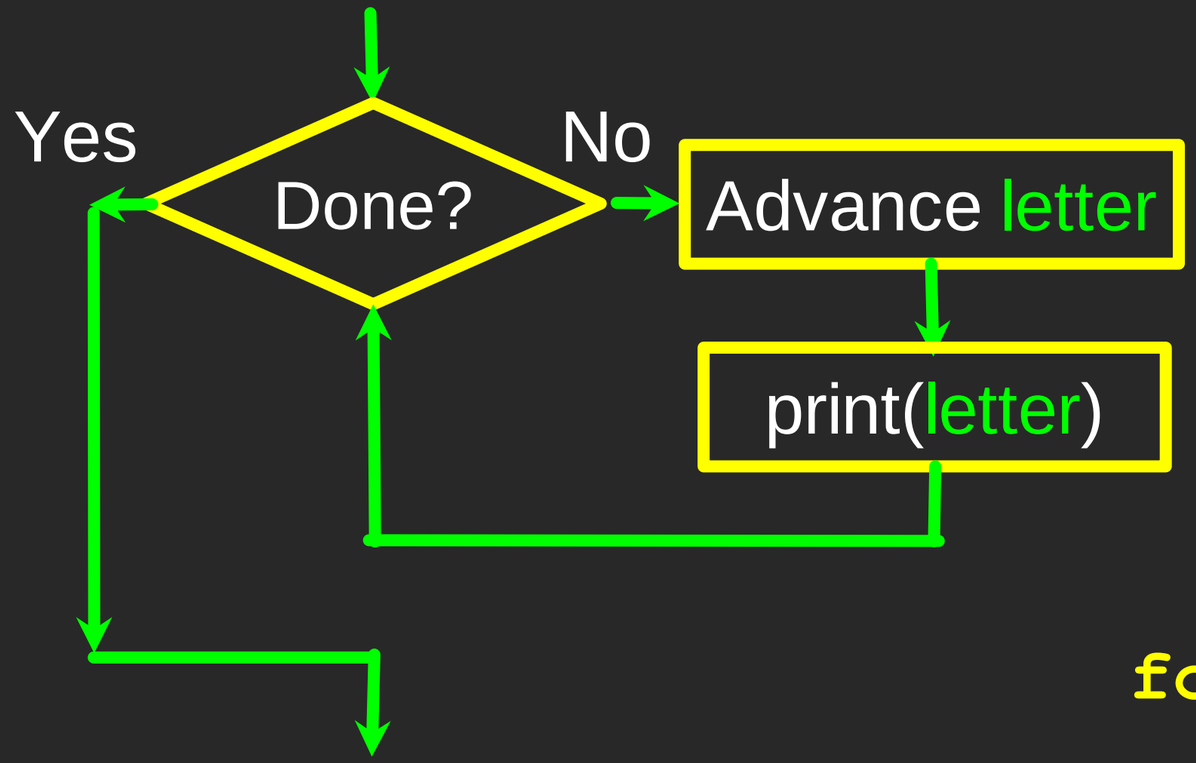
This is a simple loop that loops through each letter in a string and counts the number of times the loop encounters the 'a' character

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

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





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

Slicing Strings (1 of 2)

- We can also look at any continuous section of a string using a colon operator
- The second number is one beyond the end of the slice - “up to but not including”
- If the second number is beyond the end of the string, it stops at the end


```
>>> s = 'Monty Python'
>>> print(s[0:4])
Mont
>>> print(s[6:7])
P
>>> print(s[6:20])
Python
```

Slicing Strings (2 of 2)

If we leave off the first number or the last number of the slice, it is assumed to be the beginning or end of the string respectively

```
>>> s = 'Monty Python'
>>> print(s[:2])
Mo
>>> print(s[8:])
thon
>>> print(s[:])
Monty Python
```

Manipulating Strings..

Acknowledgements / Contributions

Initial Development: Charles Severance, University of Michigan School of Information

… Insert new Contributors and Translators here