

Strings

Chapter 6

Making Friends with Python – DI Team (22-23 May, 2017)

mrchakra@microsoft .com



Mrinal Chakraborty
Data Insights Team
Microsoft India

- ✓ **Programing:** SAS, R-Server, TensorFlow and Scala
- ✓ **Big-Data:** Cloudera Hadoop certification and Spark Ecosystem
- ✓ **Machine learning:** Logistic Regression, Neural Networks, Support vector machines, XGBoost, Classification and Association rules
- ✓ **Allied Analytics skills:** Visualisation, **Marketing** & Web analytics
- ✓ **Certifications:** PMP, Certified Scrum Master & Certified in Business analytics from **Indian School of Business** <http://www.isb.edu/cba/>

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
- Raw input numbers must be **converted** from strings

```
>>> 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
```



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

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

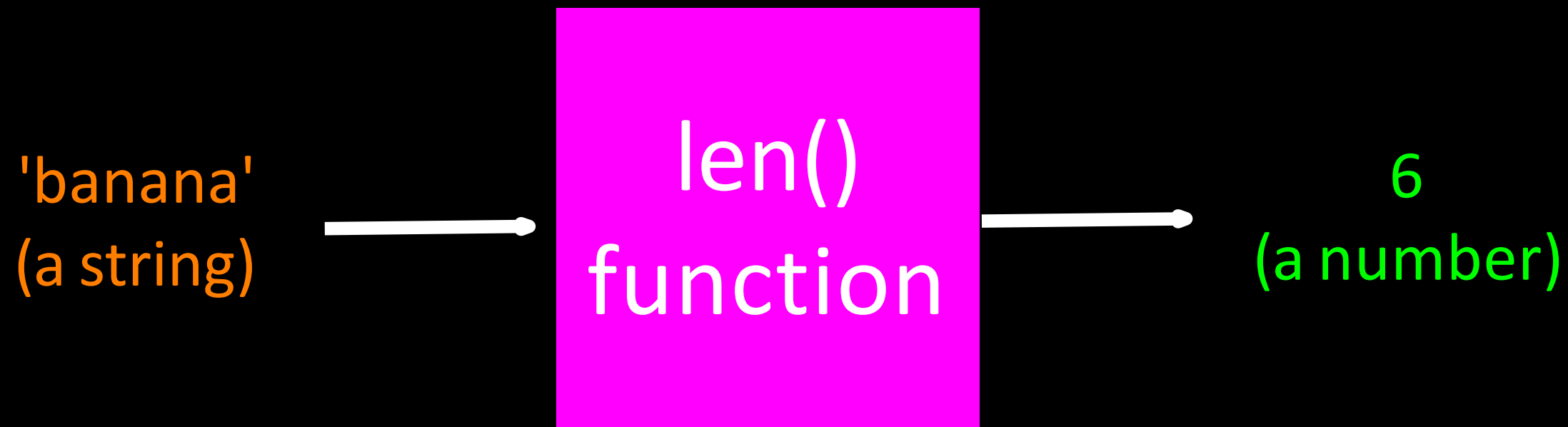
b	a	n	a	n	a
0	1	2	3	4	5

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

Len Function

```
>>> 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.



Guido wrote this code

Len Function

```
>>> 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.

'banana'
(a string)



```
def len(inp):
    blah
    blah
    for x in y:
        blah
        blah
```



6
(a number)

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'
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

- 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

- 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
```

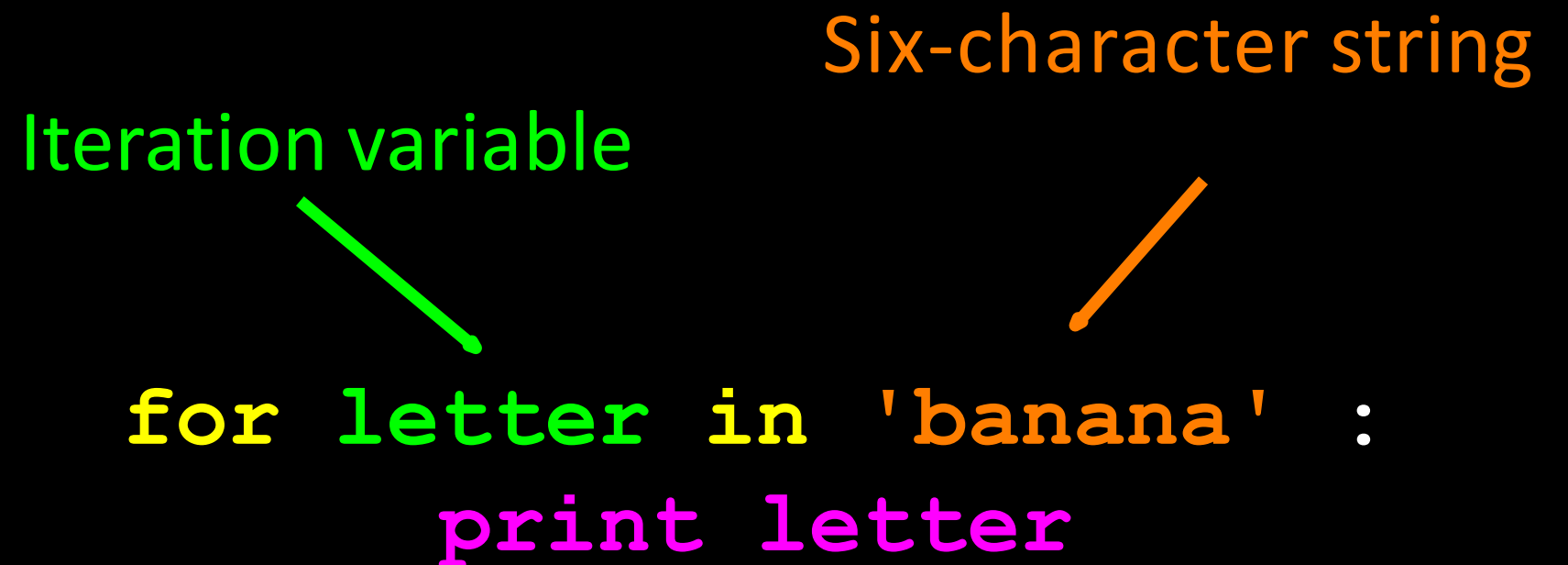
Looking deeper into **in**

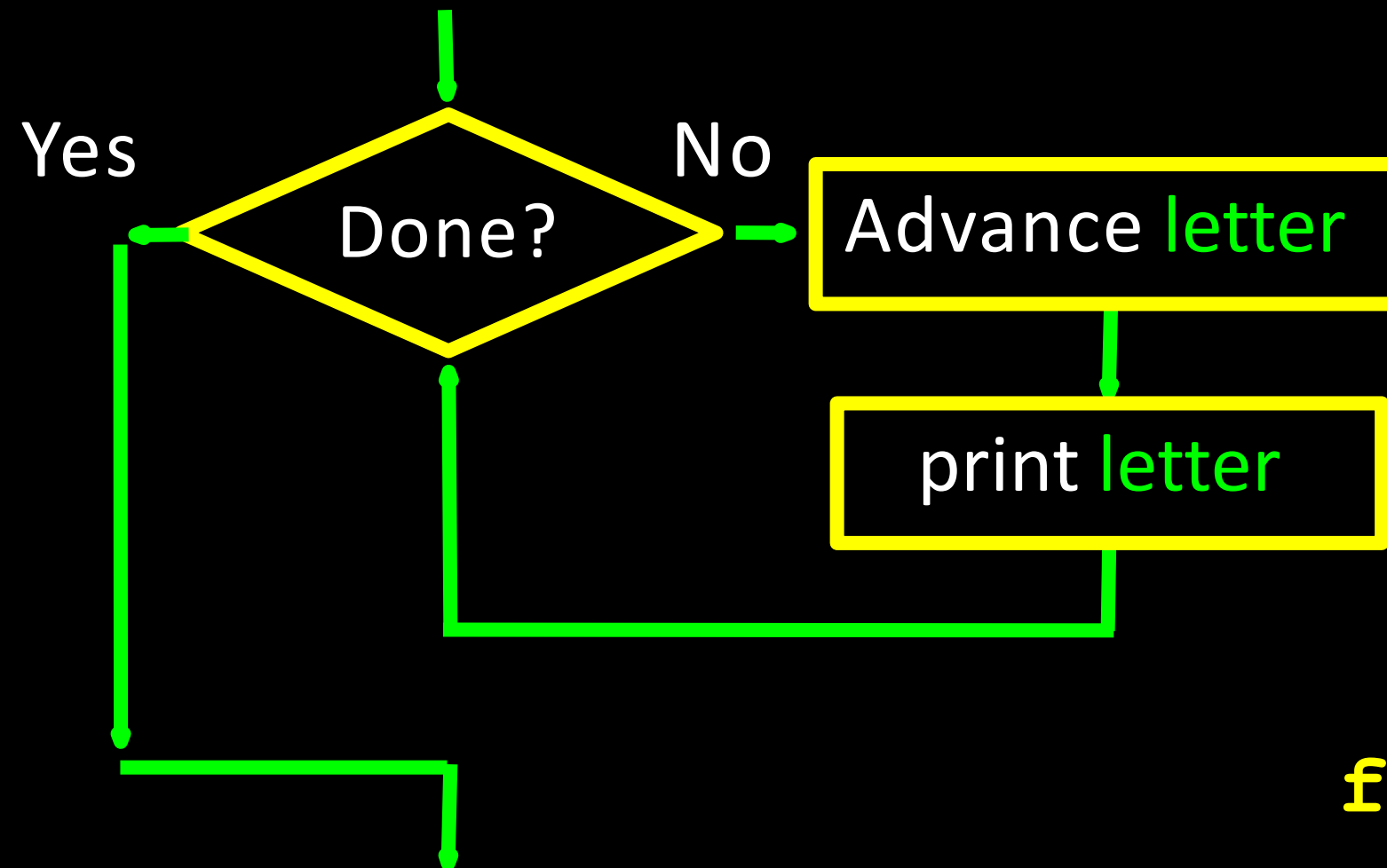
- The **iteration variable** “iterates” through the **sequence** (ordered set)
- The **block (body)** of code is executed once for each value **in** the **sequence**
- The **iteration variable** moves through all of the values **in** the **sequence**

Iteration variable

Six-character string

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





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**

M	o	n	t	y		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11

- 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

M	o	n	t	y		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11

- 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
```

Slicing Strings

String Concatenation

- When the `+` operator is applied to strings, it means “concatenation”

```
>>> a = 'Hello'
>>> b = a + 'There'
>>> print b
HelloThere
>>> c = a + ' ' + 'There'
>>> print c
Hello There
>>>
```

Using **in** as a logical Operator

- The **in** keyword can also be used to check to see if one string is “in” another string
- The **in** expression is a logical expression that returns **True** or **False** and can be used in an **if** statement

```
>>> fruit = 'banana'
>>> 'n' in fruit
True
>>> 'm' in fruit
False
>>> 'nan' in fruit
True
>>> if 'a' in fruit :
...     print 'Found it!'
...
Found it!
>>>
```

String Comparison

```
if word == 'banana':  
    print 'All right, bananas.'  
  
if word < 'banana':  
    print 'Your word,' + word + ', comes before banana.'  
elif word > 'banana':  
    print 'Your word,' + word + ', comes after banana.'  
else:  
    print 'All right, bananas.'
```

String Library

- Python has a number of string **functions** which are in the **string library**
- These **functions** are already *built into* every string - we invoke them by appending the function to the string variable
- These **functions** do not modify the original string, instead they return a new string that has been altered

```
>>> greet = 'Hello Bob'
>>> zap = greet.lower()
>>> print zap
hello bob
>>> print greet
Hello Bob
>>> print 'Hi There'.lower()
hi there
>>>
```

```
>>> stuff = 'Hello world'
>>> type(stuff)
<type 'str'>
>>> dir(stuff)
['capitalize', 'center', 'count', 'decode', 'encode',
'endswith', 'expandtabs', 'find', 'format', 'index',
'isalnum', 'isalpha', 'isdigit', 'islower', 'isspace',
'istitle', 'isupper', 'join', 'ljust', 'lower',
'lstrip', 'partition', 'replace', 'rfind', 'rindex',
'rjust', 'rpartition', 'rsplit', 'rstrip', 'split',
'splitlines', 'startswith', 'strip', 'swapcase',
'title', 'translate', 'upper', 'zfill']
```

<https://docs.python.org/2/library/stdtypes.html#string-methods>

`str.replace(old, new[, count])`

Return a copy of the string with all occurrences of substring *old* replaced by *new*. If the optional argument *count* is given, only the first *count* occurrences are replaced.

`str.rfind(sub[, start[, end]])`

Return the highest index in the string where substring *sub* is found, such that *sub* is contained within *s*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation. Return `-1` on failure.

`str.rindex(sub[, start[, end]])`

Like `rfind()` but raises `ValueError` when the substring *sub* is not found.

`str.rjust(width[, fillchar])`

Return the string right justified in a string of length *width*. Padding is done using the specified *fillchar* (default is a space). The original string is returned if *width* is less than `len(s)`.

String Library

`str.capitalize()`

`str.center(width[, fillchar])`

`str.endswith(suffix[, start[, end]])`

`str.find(sub[, start[, end]])`

`str.lstrip([chars])`

`str.replace(old, new[, count])`

`str.lower()`

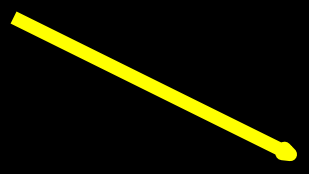
`str.rstrip([chars])`

`str.strip([chars])`

`str.upper()`

Searching a String

- We use the `find()` function to search for a substring within another string
- `find()` finds the first occurrence of the substring
- If the substring is not found, `find()` returns `-1`
- Remember that string position starts at zero



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

```
>>> fruit = 'banana'
>>> pos = fruit.find('na')
>>> print pos
2
>>> aa = fruit.find('z')
>>> print aa
-1
```


Making everything UPPER CASE

- You can make a copy of a string in **lower case** or **upper case**
- Often when we are searching for a string using **find()** - we first convert the string to lower case so we can search a string regardless of case

```
>>> greet = 'Hello Bob'
>>> nnn = greet.upper()
>>> print nnn
HELLO BOB
>>> www = greet.lower()
>>> print www
hello bob
>>>
```

Search and Replace

- The `replace()` function is like a “search and replace” operation in a word processor
- It replaces **all** occurrences of the search string with the replacement string

```
>>> greet = 'Hello Bob'
>>> nstr = greet.replace('Bob', 'Jane')
>>> print nstr
Hello Jane
>>> nstr = greet.replace('o', 'x')
>>> print nstr
Hel1x Bxb
>>>
```

Stripping Whitespace

- Sometimes we want to take a string and remove whitespace at the beginning and/or end
- `lstrip()` and `rstrip()` remove whitespace at the left or right
- `strip()` removes both beginning and ending whitespace

```
>>> greet = '    Hello Bob    '  
>>> greet.lstrip()  
'Hello Bob '  
>>> greet.rstrip()  
'    Hello Bob'  
>>> greet.strip()  
'Hello Bob'  
>>>
```

Prefixes

```
>>> line = 'Please have a nice day'
```

```
>>> line.startswith('Please')
```

```
True
```

```
>>> line.startswith('p')
```

```
False
```

Parsing and Extracting

21 31
↓ ↓
From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

```
>>> data = 'From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008'
>>> atpos = data.find('@')
>>> print atpos
21
>>> sppos = data.find(' ', atpos)
>>> print sppos
31
>>> host = data[atpos+1 : sppos]
>>> print host
uct.ac.za
```

Summary

- String type
- Read/Convert
- Indexing strings []
- Slicing strings [2:4]
- Looping through strings with **for** and **while**
- Concatenating strings with +
- String operations
- String library
- String Comparisons
- Searching in strings
- Replacing text
- Stripping white space

Acknowledgements / Contributions

These slides are Copyright 2017- The University of Michigan School of Information and open.umich.edu and made available under a Creative Commons Attribution 4.0 License. Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.