Strings, Indexing, and Slicing

Introduction to Computer Programming

Dr. Paul Vrbik

September 21, 2018

Definition (String)

Anything (with some excpetions) enclosed by single-quotes ‘ ’ or double-quotes “ ” is considered a string by Python.

A string is an ordered collection of the characters (e.g. unicode and ascii) allowed by the computer.

```
>>> "hello world"
‘hello world’
```

```
>>> type("hello world")
<class ‘str’>
```

```
>>> hello world
SyntaxError: invalid syntax
```

note the lack of quotes

```
>>> hello
NameError: name ‘hello’ is not defined
```

note the lack of quotes

Adding Strings

```
>>> "hello" + "world"
‘helloworld’


>>> type(" ")
<class ‘str’>


>>> empty_string = " "
>>> "hello" + empty_string + "world"
‘hello world’
```

Note when strings are added a new string is created.

String Equality

```
>>> "hello" == "hello"
True
```

```
>>> "hello " == "hello"
False
```

```
>>> "h e l l o" == "hello"
False
```

```
>>> "Hello" == "hello"
False
```
strings are case sensitive: `"H" != "h"`

Comparing Strings

```
>>> "a" < "b"
True
```

```
>>> "A" < "a"
True
```

```
>>> "Z" < "a"
True
```

```
>>> "a" < "aa"
True
```

```
>>> "b" < "aa"
False
```

```
>>> "aba" < "ab"
False
```

```
>>> "aZ" < "aa"
True
```

New Line

A new line or carriage return is a special escape character that can be used in strings to print what is subsequent on a new line.

The new line escape character is `\n`.

```
>>> "hello\nworld"
'hello\nworld'


>>> print("hello\nworld")
hello
world
```

Tab

A tab is a fixed amount of horizontal space. How a tab is displayed depends on the program displaying it.

(This is why tabs are the worst :)

```
>>> "hello\tworld"
'hello\tworld'
>>> print("hello\tworld")
'hello   world'
```

Note what happens when this is pasted to my IDE.

Escaping escape characters

Question

How can we print `‘‘\n\t\’’’` ?

“\\” prints a backslash “\”; “\‘” prints a single quote “ ’ ”; “\”” prints double quotes.

```
>>> x = ‘‘\\n\\’’’
>>> print(x)
\n\’
```

Numbers versus Strings

```
>>> 3 + 7
10
>>> "3" + "7"
37
>>> 3 + "7"
TypeError: unsupported operand type(s) for +: 'int'
 and 'str'
>>> str(3) + "7"
37
>>> 3 + int("7")
10
```

```
>>> 3 + int("7")
10
```

```
>>> float("123.456")
123.456
```

This is only true for numbers!

```
>>> int("hello")
ValueError: invalid literal for int() with base 10:
'hello'
```

Substitution

There is a mechanism for printing string variables in sentences through substitution.

```
>>> x = "hello"
>>> y = "world"
>>> z = "{}ooo {}ddd".format(x,y)
>>> print(z)
helloooo worldddd
```

Length

A strings length is the number of characters that comprise it.

```
>>> len("h")
1
>>> len("hello")
5
>>> x = "world"
>>> len(x)
5
>>> len(x+"world") == len(x) + len("world")
True
```

Inclusion

As a string can be regarded as an ordered set we can use the element of.

```
>>> "h" in "hello world"
True
>>> "hello" in "hello world"
True
>>> x = "world"
>>> x in "hello world"
True
>>> "ow" in "hello world"
False
```

String Indexing

Because a string is ordered we can number its characters starting from zero and access them by using square brackets.

```
>>> x = "hello world"
>>> x[0]
‘h’
>>> x[1]
‘e’
>>> x[2]
‘l’
>>> x[len(x)]
IndexError: string index out of range
```

We can also index from the end.

```
>>> x = "hello world"
>>> x[-1]
‘d’
>>> x[-2]
‘l’
>>> x[-3]
‘r’
```

String Slicing

Because the string's characters are numbered we can slice the string to obtain only a part of it.

```
>>> x = "0123456789"
```
So index matches character.

```
>>> x[1:4]
```
grab 1st inclusive through 4th exclusive characters

```
'123'
>>> x[0:9]
'012345678'
>>> x[0:10]
'0123456789'
```

```
>>> x = "0123456789"
>>> x[-1] == x[len(x)-1]
True
>>> x[3:-1]
'345678'
>>> x[3:]
'3456789'
>>> x[:]
'0123456789'
>>> x[-7:]
'3456789'
```

```
>>> x = "0123456789"
>>> x[0:-1:2]
```

grab every every 2nd character from 0th position

```
‘02468’
>>> x[1:-3:3]
‘14’
>>> x[::3]
‘0369’
>>> x[::-1]
‘9876543210’
```

We reversed the list!

```
>>> x[::-4]
‘951’
```

Immutability of Stings

Something is immutable when it cannot be changed. Strings are immutable.

```
>>> "hello"[0] = "H"
TypeError: ‘str’ object does not support item assignment
```

```
>>> x = "hello"
>>> x[0] = "H"
TypeError: ‘str’ object does not support item assignment
```

Question

Write a program that takes two strings and returns the average length of those strings.

Next Time

Question

1. If-statements. Finally!