Strings

Strings: Let’s Look Inside Them!

- Until now we’ve looked at strings as a simple Python data-type… a whole object
  - E.g., “Hello, world!”, “Fred”, “California”, “yellow”
- Now we’re going to look inside them
- Examine what they are made of
- Learn how to process their contents
  - Convert them, shorten them, reverse them, etc.

Working with Strings

- Strings are collections (sequences)
  - The characters are like list elements
  - E.g., myString[3]
- Use string index to get individual characters
  - Space (‘ ’) is a character!
  - 1st character is index [0]

```
myString = “hello world”
print(myString[1])
print(myString[6])
```

Working with Strings

- Can index string from **front** (first: [0]) or from **back** (last: [-1])

0	1	2	3	4	5	6	7	8	9	10
H	**e**	**l**	**l**	**o**		**w**	**o**	**r**	**l**	**d**
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

- len(aString): returns length
  - # of characters, ***not*** index of last character!

Strings: Comparing

- As with numbers, you can compare strings to see if equal, greater, less.

```
>>> str1 = “fred”
>>> str2 = “flintstone”
>>> str3 = “freddy”
>>> str1 == str2
>>> str1 > str2
>>> str1 <= str3
```

Strings: Concatenation & Repetition

- Concatenation: str1 + str2

```
>>> s1 = “foo”
>>> s2 = “bar”
>>> s1 + s2
```

- Repetition: str1 * n

```
>>> s3 = “hello”
>>> s4 = “goodbye”
>>> (s3 * 3) + (s4 * 2)
```

Strings:
Concatenation & Repetition

- Operator precedence: * before **+**

```
>>> s1 = “hello”
>>> s2 = “goodbye”
>>> s1 + s2 * 3
hellogoodbyegoodbyegoodbye
>>> (s1 + s2) * 3
hellogoodbyehellogoodbyehellogoodbye
```

Strings:
Built-in Boolean Functions

- str1 ***in*** str2: returns True if str2 contains str1
- str1 ***not in*** str2: opposite of ***in***

Strings are Immutable

- The contents of a string cannot be changed
  - Cannot assign new character to one string element

    ```
    myString[index] = ‘x’    #Error!
    ```

  - If it seems like you’re changing a string, you’re really making a ***new*** string and replacing the old one

```
>>> name = “Winston”
>>> name = name + “ Churchill”
```

Traversing (Looping Through) Strings

- ***for*** loops on strings

```
for ch in myString:
    print(ch*2)
    #do something
```

- Can also use ***for*** loops with index

```
for i in range(len(myString)):
    print(myString[i]*2)
    #do something
```

Traversing (Looping Through) Strings

- Can also use ***while*** loops

```
i = 0
while i < len(myString):
    print(myString[i])
    #do something
  i = i + 1
```

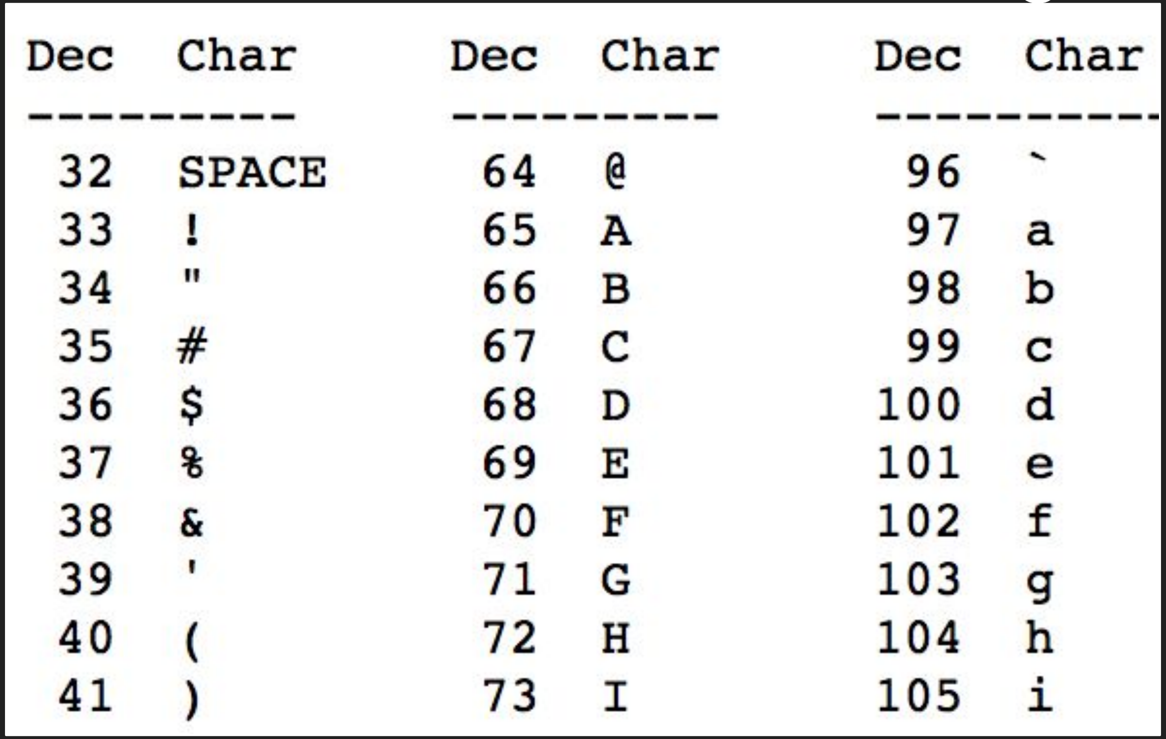
Can Slice Strings

- myString[start:end+1:step] (again Gaddis book says ‘end’)
  - Returns **new** string with characters starting with character <start> up to (***but not including***) <end>
  - If start unspecified, it’s 0
  - If end unspecified, it’s len(string)
  - If step unspecified, it’s 1
- Examples:
  - myString[1:6]: start at 1, end at 6-1= 5, step 1
  - myString[:9:2]: start at 0, end at 8, step 2
  - myString[1::2]: start at 1, end at end, step 2

Strings are Made of Characters

- Characters are represented in computers by code-numbers
- American Standard Code for Information Interchange
- ASCII table,

(note that capital letters have lower associated code numbers)

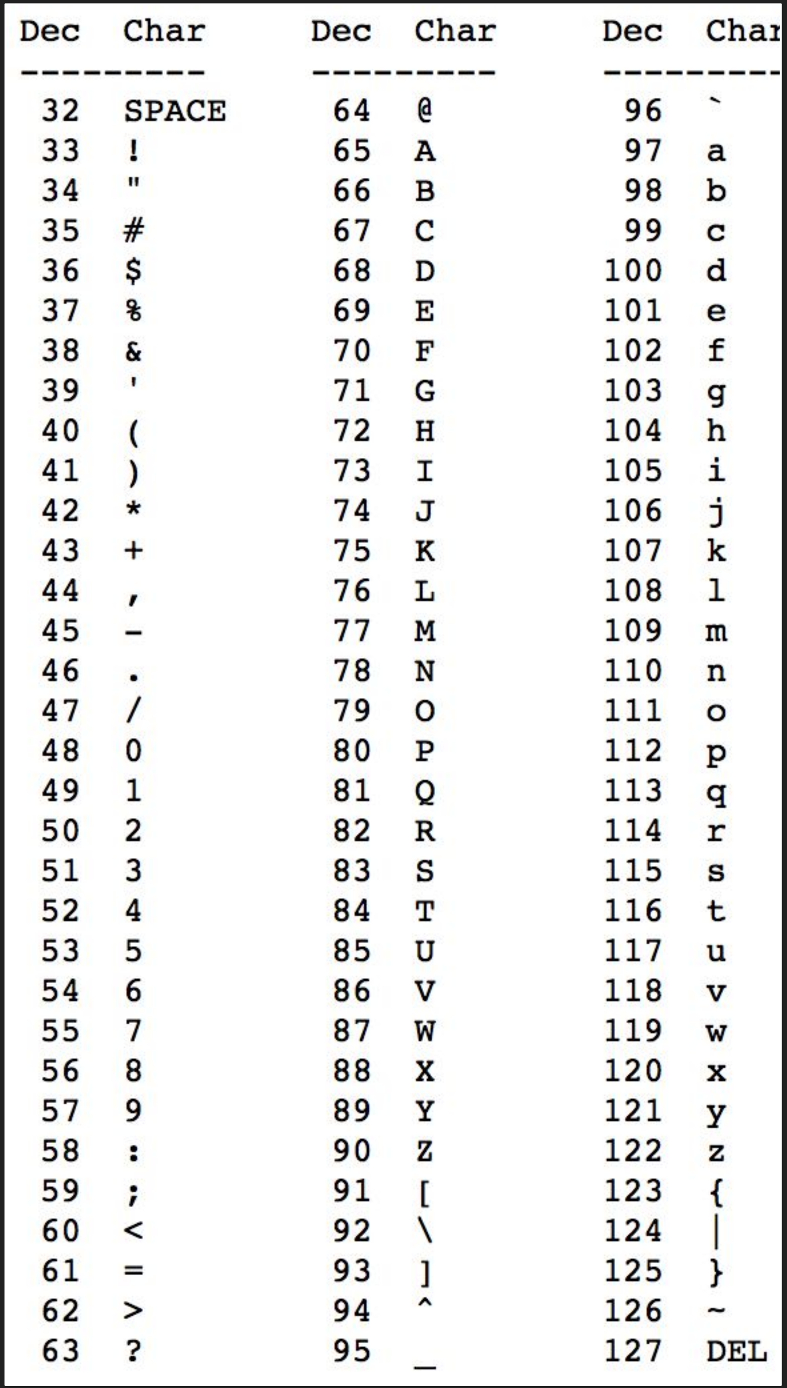


Dec	Char	Dec	Char	Dec	Char
32	SPACE	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i

ASCII Codes

- ord(c): converts char to ascii #
- chr(n): converts ascii # to char

- Test if letter is upper-case… how?
- Test if letter is lower-case… how?
- Convert lower-case letter to upper… how?
- Convert upper-case letter to lower… how?



Dec	Char	Dec	Char	Dec	Char
32	SPACE	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_	127	DEL

String Testing Methods

- <string name>.<method>(arguments)
- Boolean result: True or False

isalnum(): is the string alphanumeric?

isalpha(): is the string alphabetical?

isdigit(): is the string all numerical?

islower(): is the string all lower case?

isupper(): is the string all UPPER CASE?

isspace(): Is the string all spaces?

startswith(substr): does string start with substr?

endswith(substr): does string end with substr?

String Modification Methods

- <string name>.<method>(arguments)
- Creates ***new string*** based on given string

lower(): convert chars to lower case

upper(): convert chars to upper case

lstrip(): strip spaces from left end

rstrip(): strip spaces from right end

strip(): strip spaces from both ends

split(): splits string into list of words

split() (splits a string into a list)

```
string.split(separator)

txt = "bonjour, je m’appelle Beste, j’ai douze ans"

x = txt.split(", ")

print(x)

['bonjour', 'je m’appelle Beste', 'j’ai douze ans']
```

String Search Methods

- <string name>.<method>(arguments)

find(substr): index where substring starts

- Or -1 if not found

rfind(substr): rightmost index of substring

- Or -1 if not found

replace(old, new): swap new for old in new copy of string

```
>>> old_string='la di da'
>>> new_string=old_string.replace('la', 'laaaaaa')
>>> new_string
'laaaaaa di da'
```

String Accumulators

- Number accumulators (we already saw)

```
count = 0
totalExpenses = 0
<Some kind of loop>:
    count = count + 1
    totalExpenses = totalExpense + newExpense
```

- String accumulators

```
nameString = “”            # empty (null) string
<Some kind of loop>:
    newName = input(”Enter next name: “)
    nameString = nameString + “,” + newName
```

String Accumulators

- Copying a string (by item)

```
str1 = “truth cannot wait”
str2 = “”
for c in str1:
   str2 = str2 + c
print(str2)
```

- Copying a string, 2nd way (by index)

```
str1 = “truth cannot wait”
str2 = “”
for i in range(len(str1)):
   str2 = str2 + str1[i]
```

String Accumulators

- Reversing a string: how to code it with loops?

  str1 = “truth cannot wait”

  str3 = “”

  Try reversing str1 by using a for loop by item and concatenating with str3