Review of the base operators for strings

- myStr[3]
- myStr[3:6]
- Addition
- Multiplication
- in

Another Operator

- Can check to see if a substring exists in the string using the `in` operator.
- Returns True or False

```
myStr = ‘aabbccdd’
 ‘a’ in myStr ⇒ True
 ‘abb’ in myStr ⇒ True
 ‘x’ in myStr ⇒ False
```

Functions

- What is a function?

- Name several we have worked with.

Functions, First Cut

- A function is a program that performs some operation(s). Its details are hidden (encapsulated), only its interface provided.
- A function takes some number of inputs (arguments) and returns a value based on the arguments and the function’s operation.

String Method

- A **method** is a variation on a function
    - like a function, it represents a program
    - like a function, it has input arguments and an output
- Unlike a function, it is applied in the context of a particular object.
- This is indicated by the ‘dot notation’ invocation

Example

- upper is the name of a method. It generates a new string that has all upper case characters of the string it was called with.

myStr = ‘Python Rules!’

myStr.upper() ⇒ ‘PYTHON RULES!’

More Dot Notation

- Dot notation looks like this:
  - object.method(…)
- It means that the object in front of the dot is calling a method that is associated with that object’s type.
- The methods that can be called are tied to the type of the object calling it. Each type has different methods.

Find

```
myStr = ‘hello’
myStr.find(‘l’)          # find index of ‘l’ in myStr
 ⇒ 2
```

Note how the method ‘find’ operates on the string object myStr and the two are associated by using the “dot” notation: myStr.find(‘l’).

Terminology: the thing(s) in parenthesis, i.e. the ‘l’ in this case, is called an **argument**.

Chaining Methods

Methods can be chained together.

- Perform first operation, yielding an object
- Use the yielded object for the next method

myStr = ‘Python Rules!’

myStr.upper() ⇒ ‘PYTHON RULES!’

myStr.upper().find(‘O’)

⇒ 4

Optional Arguments

Some methods have optional arguments:

- if the user doesn’t provide one of these, a default is assumed
- find has a default second argument of 0, where the search begins

```
aStr = ‘He had the bat’
aStr.find(‘t’) ⇒ 7 # 1st ‘t’,start @ 0
aStr.find(‘t’,8) ⇒ 13 # 2nd ‘t’
```

Nesting Methods

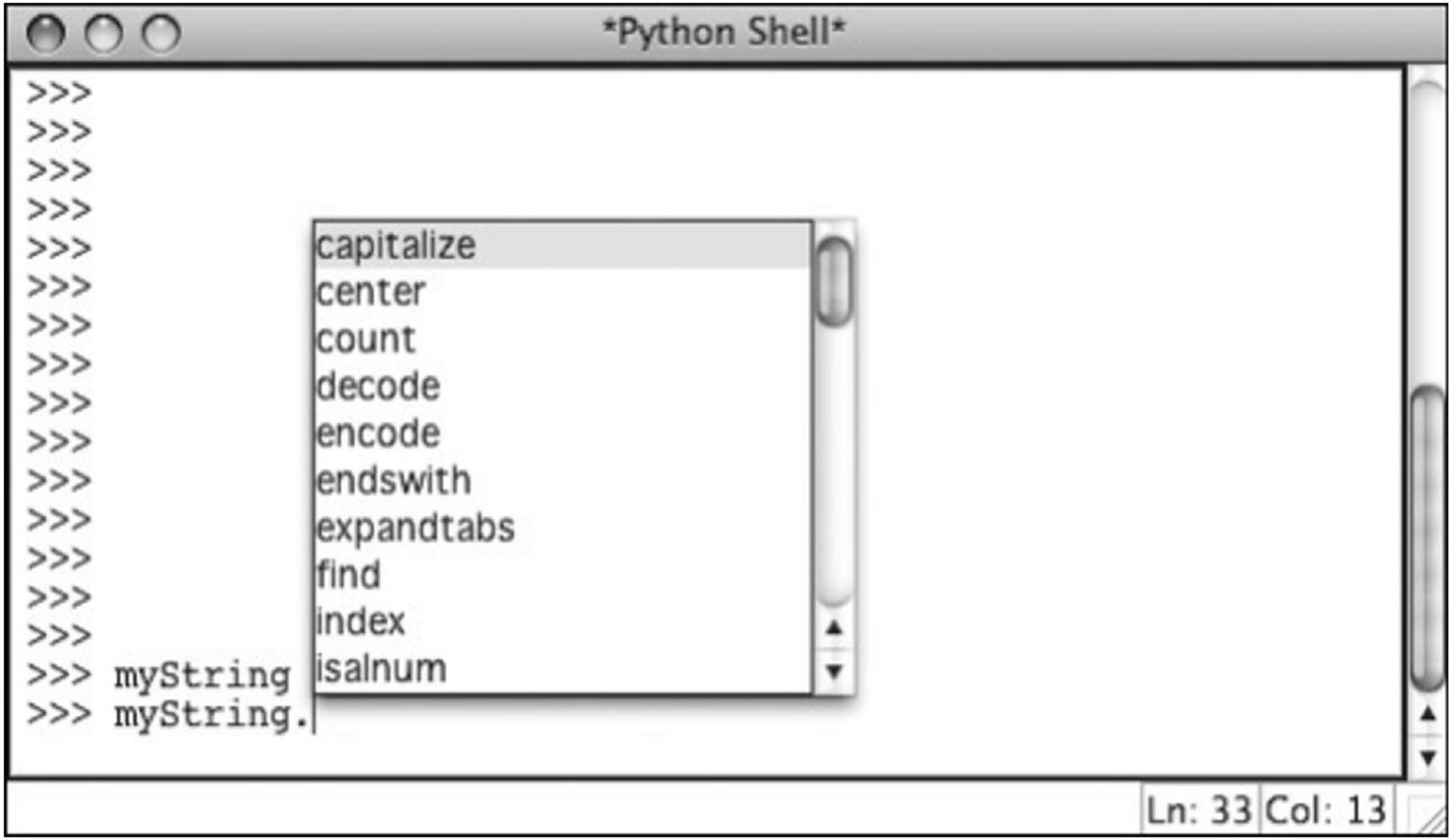
- You can “nest” methods, that is, the result of one method as an argument to another.
- Remember that parenthetical expressions are done “inside out”: do the inner parenthetical expression first, then the next, using the result as an argument.

```
aStr.find(‘t’, aStr.find(‘t’)+1)
```

- Translation: find the second ‘t’.

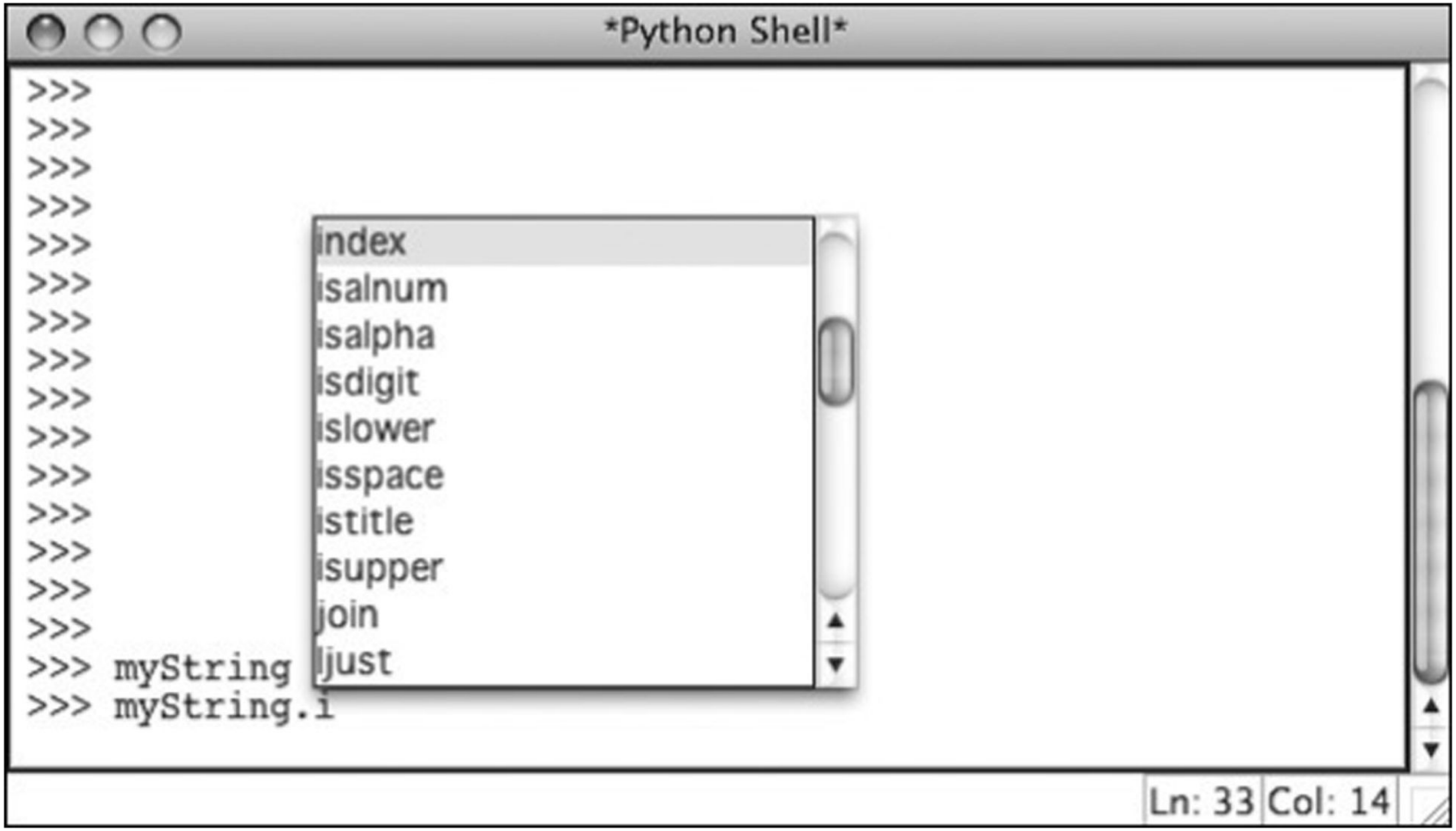
How to Know?

- You can use IDLE to find available methods for any type. You enter a variable of the type, followed by the ‘.’ (dot) and then a tab.
- Remember, methods match with a type. Different types have different methods.
- If you type a method name, IDLE will remind you of the needed and optional arguments.



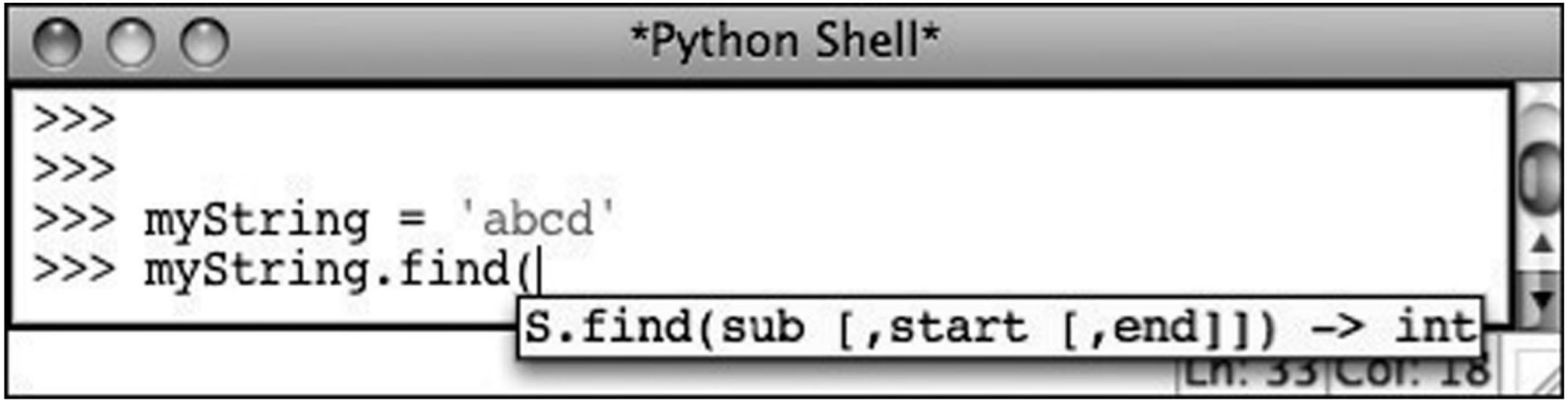
```
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>> myString
>>> myString.
```

FIGURE 4.7 In IDLE, tab lists potential methods.



```
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>> myString
>>> myString.i
```

FIGURE 4.8 In IDLE, tab lists potential methods, with leading letter.



Python Shell

```
>>>
>>>
>>> myString = 'abcd'
>>> myString.find(
S.find(sub [,start [,end]]) -> int
```

FIGURE 4.9 IDLE pop-up provides help with function arguments and return types.

More Methods

(Even more exist: http://docs.python.org/lib/string-methods.html)

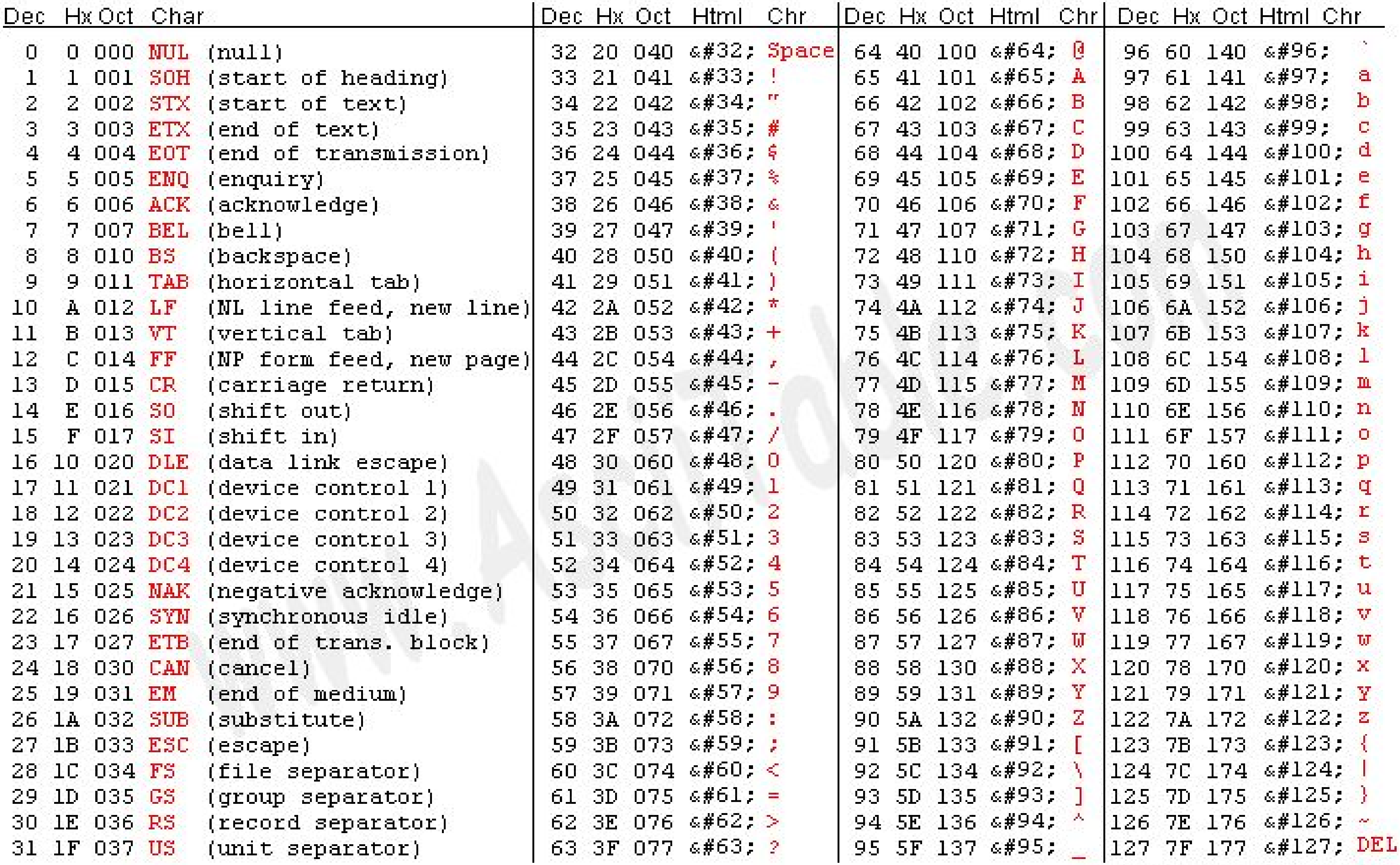
- s.capitalize
- s.center(width)
- s.count(sub,[,start [,end]])
- s.ljust(width)
- s.lower()
- s.upper()
- s.lstrip()
- s.rfind(sub, [,start [,end]])
- s.splitlines([keepends])
- s.strip()
- s.translate(table [, delchars])

String Comparisons, Single Char

- There are multiple systems for representing characters: ASCII, Unicode, windows-1252, etc.
- ASCII takes the English letters, numbers and punctuation marks and associates them with an integer number (0-128, or 256 for extended set)
- Single character comparisons are based on that number

String Encodings

- We can get the encodings from characters using the ord function
  - >>> ord('x')
  - Humans can look this number up in the ASCII table
- We can get the characters back from the encoding using the chr function
  - >>>chr(120)



Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH (start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX (start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX (end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT (end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ (enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK (acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL (bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS (backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB (horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF (NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT (vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF (NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR (carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO (shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI (shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE (data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1 (device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2 (device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3 (device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4 (device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK (negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN (synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB (end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN (cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM (end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB (substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC (escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS (file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS (group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS (record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US (unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

Source: www.asciitable.com

Comparisons Within Sequence

- It makes sense to compare within a sequence (lower case, upper case, digits).
  - ‘a’ < ‘b’   True
  - ‘A’ < ‘B’   True
  - ‘1’ < ‘9’   True
- Can be weird outside of the sequence:
  - ‘a’ < ‘A’   False
  - ‘a’ < ‘0’   False
- … because we are really comparing the ord() encodings of each character

Whole Strings

- Compare the first element of each string:
  - if they are equal, move on to the next character in each
  - if they are not equal, the relationship between those to characters are the relationship between the string
  - if one ends up being shorter (but equal), the shorter is smaller

Examples

- ‘a’ < ‘b’ True
- ‘aaab’ < ‘aaac’
  - First difference is at the last char. ‘b’<‘c’ so ‘aaab’ is less than ‘aaac’. True.
- ‘aa’ < ‘aaz’
  - The first string is the same but shorter. Thus it is “smaller”. True.

Penny Math

- Penny Math is a simple formula
  - A (or a) costs 1 penny
  - B (or b) costs 2 pennies
  - …
  - Z (or z) costs 26 pennies
  - Everything else is FREE
- Thus
  - "Sergey" costs 19+5+18+7+5+25=79 cents

Our next task

- Write a program called pennyMath that reads in a String and prints the integer value corresponding to the “cost” of the String.
  - Version a: uses an “alphabet” string
  - Version b: uses the ord() function instead