Chapter 3, Miller 3rd ed, Codes and Other Secretes

Section 3.2, Miller 3rd ed, The String Data Type

1. More on Strings

a. *Concatenation*

```
name = 'Bob'
age = 22
message = name + ' is ' + str(age) + ' years old'
```

→ 'Bob is 22 years old'

Easier:

```
message = f'{name} is {age} years old'
```

Good Concatenation Example

```
firstName = 'Ron'
lastName = 'Obvious'
fullName = firstName + " " + lastName
```

b. *Repetition*

`'go' * 3` → 'gogogo'

c. *Indexing*

'Davenport'

0	1	2	3	4	5	6	7	8
D	a	v	e	n	p	o	r	t
-9	-8	-7	-6	-5	-4	-3	-2	-1

```
town = 'Davenport'
town[0]          → D
town[9]          → Error
town[-1]         → t, the last character
```

1st char always at	[0]
Last char always at	[-1]

d. *Length of a string*

`len(town)` → 9

Valid index i of a string s

-len(s) ≤ i ≤ len(s) - 1

e. Iterating over a string - Two ways

i. Using the index

```
for i in range(len(town)):
    print(town[i])
```

ii. Using string iteration

```
for ch in town:
    print(ch)
```

03-01-printChars.py

`'hotdog'` is printed as

h	, o	, t	, d	, o	, g

f. Containment, in and not in

```
town = 'Davenport'
```

`'port' in town` → True

`'city' in town` → False

`'port' not in town` → False

`'city' not in town` → True

g. Slicing a string - Get a substring

Slice Operator `[ i : j : k ]`

i = Start	Default 0
j = Stop (one after)	Default len(string)
k = Step	Dafault 1

0	1	2	3	4	5	6	7	8
D	a	v	e	n	p	o	r	t
-9	-8	-7	-6	-5	-4	-3	-2	-1

Slice Example 1 - Basic substring example

```
town = 'Davenport'
```

`town[ 5 : 8 ]` → 'por'

0	1	2	3	4	5	6	7	8
D	a	v	e	n	p	o	r	t
-9	-8	-7	-6	-5	-4	-3	-2	-1

Slice Example 2 - Start is omitted. 0 is used

```
town = 'Davenport'
town[ : 8 ]  → 'Davenpor'
town[ 0 : 8 ]  → 'Davenpor'
```

0	1	2	3	4	5	6	7	8
D	a	V	e	n	p	o	r	t
-9	-8	-7	-6	-5	-4	-3	-2	-1

Slice Example 3 - Stop is omitted. Length of string is used

```
town = 'Davenport'
town[ 5 : ]  → 'port'
town[ 5 : len(town) ]  → 'port'
```

0	1	2	3	4	5	6	7	8
D	a	V	e	n	p	o	r	t
-9	-8	-7	-6	-5	-4	-3	-2	-1

Slice Example 4 - Start and Stop are both omitted

```
town = 'Davenport'
town[ : ]  → 'Davenport'
Same as:
town[ 0 : len(town) ]  → 'Davenport'
```

0	1	2	3	4	5	6	7	8
D	a	v	e	n	p	o	r	t
-9	-8	-7	-6	-5	-4	-3	-2	-1

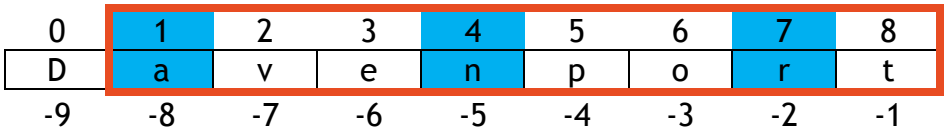
Slice Example 5 - Set step to 2 to get every other character

```
town = 'Davenport'
town[ : : 2 ]  → 'Dvnot' (Start at 0, go to len(town)-1, step by 2)
```

0	1	2	3	4	5	6	7	8
D	a	v	e	n	p	o	r	t
-9	-8	-7	-6	-5	-4	-3	-2	-1

Slice Example 6 - Set step to 3

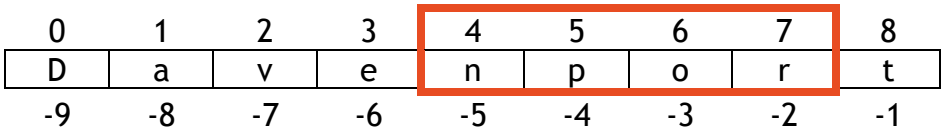
```
town = 'Davenport'
town[ 1 : : 3 ]
```
→ 'anr' (start at 1, go to len(town)-1, step by 3)



0	1	2	3	4	5	6	7	8
D	a	v	e	n	p	o	r	t
-9	-8	-7	-6	-5	-4	-3	-2	-1

Slice Example 7 - Negative Step

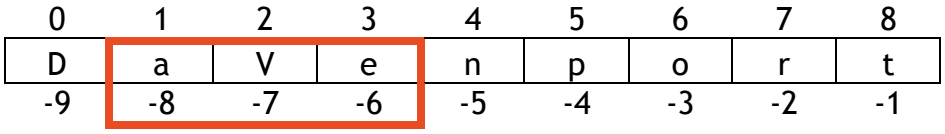
```
town = 'Davenport'
town[ 7 : 3 : -1 ]
```
→ 'ropn'



0	1	2	3	4	5	6	7	8
D	a	v	e	n	p	o	r	t
-9	-8	-7	-6	-5	-4	-3	-2	-1

Slice Example 8 - Negative Indexes

```
town = 'Davenport'
town[ -8 : -5 : ]
```
→ 'ave'

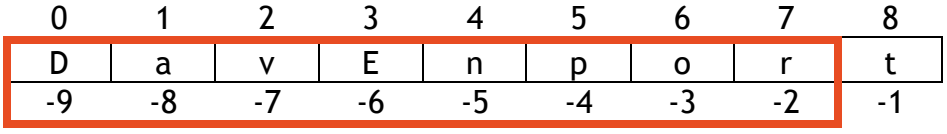


0	1	2	3	4	5	6	7	8
D	a	V	e	n	p	o	r	t
-9	-8	-7	-6	-5	-4	-3	-2	-1

Slice Example 9 - Getting the Whole String

Can't use negative index to get the whole string. Leaves off last character

```
town = 'Davenport'
town[ -9 : -1 ]
```
→ 'Davenpor'



0	1	2	3	4	5	6	7	8
D	a	v	E	n	p	o	r	t
-9	-8	-7	-6	-5	-4	-3	-2	-1

Going to 0 doesn't work, 0 is the first letter

```
town = 'Davenport'
town[ -9 : 0 ]
```
→ ''

*Must use **positive index** to get the **whole string***

```
town = 'Davenport'
town[ 0 : 9 ]
```
→ 'Davenport'

0	1	2	3	4	5	6	7	8
D	a	v	e	n	p	o	r	t
-9	-8	-7	-6	-5	-4	-3	-2	-1

Slice Example 9 - Reversing the string

Similar problem as getting the whole string

Can't use positive index to reverse the string, leaves off 1st character

```
town = 'Davenport'
town[ 8 : 0 : -1]
```
→ 'tropneva'

0	1	2	3	4	5	6	7	8
D	a	v	E	n	p	o	r	t
-9	-8	-7	-6	-5	-4	-3	-2	-1

Can't go down to -1, that's the index for the last character

```
town = 'Davenport'
town[ 8 : -1 : -1]
```
→ ''

*Must use **negative indexes** to **reverse a string***

```
town = 'Davenport'
town[ -1 : -10 : -1]
```
→ 'tropnevaD '

0	1	2	3	4	5	6	7	8
D	a	v	e	n	p	o	r	t
-9	-8	-7	-6	-5	-4	-3	-2	-1

More common

```
town = 'Davenport'
town[ :  : -1]
```
→ 'tropnevaD '

1. Functions vs. Methods

   a. Built-in Python functions

      We've seen many:
      ```
      dir()  id()  input()  int()  str()  sum()  float()
      print()  len()  type()  list()  range()  complex()
      ```

   b. User-defined functions – We've written many of these

   c. Methods
      i. Similar to functions
      ii. Defined for a class or an object

2. str vs. string
   str → built-in
   string → need to import

3. str methods (built-in, don't need to import)
   Strings are ***Immutable*** – no string method **can ever** change a string

   a. center / ljust / rjust
      ```
      state = 'Iowa'

      width = 10

      state.center(width)     →      ' Iowa  '

      state.ljust(width)      →      'Iowa    '

      state.rjust(width)      →      '    Iowa'
      ```

   b. count
      ```
      state = 'Mississippi'

      state.count('i')        →   4

      state.count('ss')       →   2

      state.count('Iowa')     →   0

      state.count('issi')     →   1, uses middle 'i' only once
      ```

c. upper / lower / swapcase / capitalize / title

```
book = 'A Tale of two Cities'
```

```
book.upper()        → 'A TALE OF TWO CITIES'

book.lower()        → 'a tale of two cities'

book.swapcase()     → 'a tALE OF TWO cITIES'

book.capitalize()   → 'A tale of two cities'

book.title()        → 'A Tale Of Two Cities'
```

d. replace

```
river = 'Mississippi'

river.replace('issipp', 'our')   → 'Missour'. river not changed
```

e. find / index

Both find **first occurrence** of one string within another string

```
river = 'Mississippi'

river.find('ss')    → 2

river.index('ss')   → 2
```

They differ when it's not found
find returns -1
index raises an exception

```
river.find('dog')   → -1

river.index('dog')  → ValueError: substring not found
```

f. rfind / rindex
Like find / index, but finds ***last occurrence***

```
river = 'Mississippi'
```

```
river.rfind('ss')      →   5
river.rindex('ss')     →   5
river.rfind('dog')     →   -1
river.rindex('dog')    →   ValueError: substring not found
```

g. startswith / endswith

```
quote = 'Peace starts with a smile'
```

```
quote.startswith('Peace')              →   True
quote.startswith('peace')              →   False
quote.lower().startswith('peace')      →   True
quote.endswith('smile')                →   True
```

h. expandtabs – Creates a *new string* with tabs expanded
`print` command prints expanded tabs but doesn't save new string.

```
header = 'date\tsales'
```

```
header.expandtabs()          →   'date    sales'    default 8 spaces
header.expandtabs(12)        →   'date        sales'  change default
var = header.expandtabs()    →   var equals 'date      sales'
```

i. isalpha
Returns: True if all characters are alphabet
False if 1 or more characters are non-alphabet

```
name = 'winston churchill'
name.isalpha()                  → False     space not alpha

name = 'churchill'
name.isalpha()                  → True

number = '0xfe'
number.isalpha()                → False     0 not alpha
```

j. isdigit
Returns: True if all characters are digits
False if 1 or more characters are non-digit

```
book = '1984'
book.isdigit()                  → True

dates = '1993-2000'
dates.isdigit()                 → False     - is non-digit
```

k. isalnum
Returns: True if all characters are alpha or digit
False if 1 or more characters are not either alpha or digit

```
name = 'winston churchill'
name.isalnum()                  → False     space not alpha-num

name = 'churchill'
name.isalnum()                  → True

book = '1984'
book.isalnum()                  → True

number = '0xfe'
number.isalnum()                → True
```

l. islower / isupper

```
book = '2001: a space odyssey'
book.islower()          → True      Only looks at letters
book.isupper()          → False

book = '2001: A Space Odyssey'
book.islower()          → False     Has mixed case
book.isupper()          → False

gear = 'SCUBA'
gear.islower()          → False
gear.isupper()          → True
```

Example usage – count lowercase letters in a string
See 03-02-countLower.py

m. istitle

```
book = 'Pebble in the Sky'
book.istitle()          → False

book = 'PEBBLE IN THE SKY'
book.istitle()          → False

book = 'Pebble In The Sky'
book.istitle()          → True
```

n. isspace – checks for all white space

```
message = '   \n  \t  \t  \n'
message.isspace()       → True
```

o. zfill – Left fill with zeros

```
amount = '15.32'
amount.zfill(10)        → '0000015.32'   width is parameter
```

p. lstrip / rstrip / strip

```
book = '    \t The Manchurian Candidate\n'
book.lstrip()                  →   'The Manchurian Candidate\n'
book.rstrip()                  →   '   \t The Manchurian Candidate'
book.strip()                   →   'The Manchurian Candidate'
```

Remember these for when we read from files.

With an argument - Chars removed until we get to a string char not specified

```
river = 'Mississippi'

river.strip('M')          → 'ississippi'
river.strip('pi')         → 'Mississ'
river.strip('Mis')        → 'pp'
river.strip('Mpis')       → ''

river.rstrip('pi')        → 'Mississ'
river.lstrip('Mi')        → 'ssissippi'
```

4. The string module (need to import string)

```
import string
string.ascii_letters
   →    'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'

string.ascii_lowercase
   →    'abcdefghijklmnopqrstuvwxyz'

string.ascii_uppercase
   →    'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

string.digits
   →    '0123456789'

string.hexdigits
   →    '0123456789abcdefABCDEF'

string.octdigits
   →    '01234567'

string.punctuation
   →    '!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

```
string.whitespace
   →    ' \t\n\r\x0b\x0c'
   \t   →   Tab
   \r   →   Carriage return
   \n   →   New line
   \x0b →   Vertical tab
   \x0c →   Form feed
```