

String indexing, slicing, and concatenation

1. Write a program called **phone()** that accepts a ten-digit phone number as a string of digits and prints it out in the standard style (*nnn nnn-nnnn*), with no spaces around the hyphen or parentheses. Hint: to avoid extra spaces, use the concatenation operator `+` to construct the complete phone-number string, and then print out the new string.

```
>>> phone()
Enter a phone number: 9143952673
(914) 395-2673
```

2. Write a program called **alpha()** that asks the user for a number from 1 to 26 and prints the corresponding letter of the alphabet, exactly as shown below. Define the alphabet string below in your program and use it as a “lookup table”. Remember that “a” is at position 0 of the string, not position 1.

```
alphabet = "abcdefghijklmnopqrstuvwxyz"

>>> alpha()
Enter a number from 1 to 26: 26
Letter number 26 is z
```

3. Write a program called **numname()** that asks the user for a number from 0 to 9 and prints the corresponding number name “zero” to “nine”. Indexing in Python works with lists as well as strings, so you should use the *list* of strings below as a “lookup table”. Hint: “zero” is at position 0 of the list, “one” is at position 1, and so on.

```
numbers = ["zero", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine"]

>>> numname()
Enter a number from 0 to 9: 7
You entered seven
```

4. Now write **numname2()** to handle 2-digit numbers from 20 to 99. First, break the number into its left and right digits using the `//` and `%` operators. You can then retrieve the appropriate word from the new list below based on the left digit. Use an `if/else` statement to check if the right digit is 0, in order to correctly handle numbers like 20, 30, 40, etc., but don't worry about numbers below 20. Make sure that no spaces appear around the hyphen in your output.

```
multiples = ["twenty", "thirty", "forty", "fifty", "sixty", "seventy", "eighty", "ninety"]

>>> numname2()
Enter a number from 20 to 99: 42
You entered forty-two
```

5. Write a program called **double()** that asks the user for a string and prints out a new string with each character doubled. Hint: use a for-loop to step through the string one character at a time, building up a new string in a separate variable as you go. When the loop is done, print out the new string. For example:

```
>>> double()
Enter a string: watermelon
waaatteerrrmeelloonn
```

6. Write a program called **stretch()** that asks the user for a string and prints out a new string containing one copy of the first character, two copies of the second character, three copies of the third character, and so on. Hint: the string repetition operator `*` will come in handy here. For example:

```
>>> stretch()
Enter a string: yeow!
yeeooowww!!!!
```

Using the string `find` method

7. Write a program called `initials()` that asks the user to enter their name as a string in the format *Lastname, Firstname MI* and then prints out their initials. Hint: use the string's `find` method to determine the position of the comma, and then add 2 to get the starting position of *Firstname*. For example:

```
>>> initials()
Enter your name as Last, First MI: Marshall, James B
Your initials are JBM
```

8. Write a program called `url()` that asks the user for a URL (a web address) and then prints out the URL's *protocol*, *domain*, and *filename*. The protocol is everything up to the first `://`, the domain is everything between `://` and the following `/`, and the filename is everything after the last `/` to the end of the string. Hint: it may help to create a new temporary string with the initial protocol and `://` removed. For example:

```
>>> url()
Enter a URL: http://science.slc.edu/jmarshall/courses/fall/info.html
Protocol is http
Domain is science.slc.edu
Filename is info.html
```

Using the string `split` method

Type in the program below and try it out. This program “splits” a phrase entered by the user into a *list* of its individual words and then prints out the words one per line. Note the new form of the for-loop, without `range`. Test this program on “the rain in Spain” and a few other input phrases. Do you see how this type of for-loop works?

```
def words():
    phrase = input("Enter a phrase: ")
    wordlist = phrase.split()
    print("wordlist is", wordlist)
    print("First word:", wordlist[0])
    print("Last word:", wordlist[-1])
    for w in wordlist:
        print(w)
```

9. Using the above program as a guide, write a program called `countwords()` that asks the user for a phrase and reports the number of words in the phrase, as shown below:

```
>>> countwords()
Enter a phrase: the rain in Spain
That phrase has 4 words
```

10. Write a program called `avglen()` that asks the user for a phrase and then prints out the average length of the words in the phrase. For example:

```
>>> avglen()
Enter a phrase: the rain in Spain
The average word length is 3.5
```

11. Write a program called `acro()` that asks the user for a phrase and then constructs and prints an acronym for the phrase. The acronym should be capitalized. For example:

```
>>> acro()
Enter a phrase: federal intelligence bureau
FIB
```

12. Write a program called `revphrase()` that asks the user for a phrase, constructs a new phrase consisting of the words in reverse order, and prints out the new phrase on a single line, as shown below:

```
>>> revphrase()
Enter a phrase: the rain in Spain
Spain in rain the
```