Team #:_____ Name:_____
Absent:

1. The "print" statement can be used to write to standard output [or an optional file]. The syntax of the print statement is:

print [>> fileobject,] [arg1 [, arg2]*] [,]

A space separates each argument in the output. A new-line is added to the end unless a comma occurs at the end of the statement.

The built-in function "input" allows for program interaction with a user. The syntax the input statement is:

myInput = **input**(['prompt'])

This prints the prompt if supplied, reads the keyboard, and evaluates it. For example,
age = input("How old are you? ")
test1, test2 = input("Enter two scores to average: ")

Write Python code to read the height and width of a rectangle and prints a rectangle using askerisks ('*') with these dimensions. e,g., input of 3, 5:

```
* * * * *
* * * * *
* * * * *
```

2. List comprehensions are allows you to create one list from another based on iteration and selection criteria. For example, if you want to generate a list of squares for odd integers between 1 and 10:

```
oddSquareList = [x * x for x in range(1, 10) if x%2 ==1]
```

Use list comprehension to generate a list of tuples containing words and word lengths from a list of words, say wordList = "this is a sentence containing some words".split(' ')

3. The item iterator (iteritems( )) is a useful looping techniques when processing a whole dictionary. The

phoneNumbers = {'fienup':35918, 'gray':35917, 'east':32939,'drake':35811,'schafer':32187}

dictionary can be iterated over as in:

```
for name, number in phoneNumbers.iteritems( ):
    print  name, "has phone number", number
```

```
gray has phone number 35917
east has phone number 32939
schafer has phone number 32187
drake has phone number 35811
fienup has phone number 35918
```

(Note: Methods iteritems( ) and iterkeys( ) also exist.)

Similarly, the enumerate( ) function can be used to loop through a sequence (list, string, or tuple) to retrieve the value and index of items. For example,

```
for index, char in enumerate('cat'):
    print "'", char, "' is at", index
```

```
' c ' is at 0
' a ' is at 1
' t ' is at 2
```

Team #:_____ Name:_____
Absent:

Another useful technique that allow you to loop over "parallel" sequences at the same time is zip( ). For example,

```
names = ['fienup', 'gray', 'east', 'drake','schafer']
numbers = [35918, 35917, 32939, 35811, 32187]
for name, number in zip(names, numbers):
    print name[0].upper()+name[1:],"has phone number",number
```

Fienup has phone number 35918
Gray has phone number 35917
East has phone number 32939
Drake has phone number 35811
Schafer has phone number

Write code to iterate over a string and print the index of each of the vowels.

4. Many programs exist that do nothing more than process files of data, so file usage is important. Below is a summary of the important file operations in Python.

File Operations in Python

General syntax	Example	Description
open(filename) open(filename, mode)	`f = open('data.txt', 'w')`	Modes: `'r'` read only; `'w'` write only; `'a'` append; `'r+'` both reading and writing. On Windows and Macs, 'b' appended to the mode opens the file in binary mode. Default mode is `'r'`
f.read( )	`all = f.read()`	Returns the whole file as a string.
f.read(size)	`chunk = f.read(100)`	Returns a string of at most 100 (size) bytes. If the file has been completely read, an empty string is returned.
f.readline()	`nextLine = f.readline()`	Returns the next line from the file. The newline ('\n') character is left at the end of the string, unless it is the last line of a file which does not end in a newline character.
f.readlines()	`allLines = f.readlines()`	Returns a list containing all the lines of the file.
f.readlines(size)	`someLines = f.readlines(5000)`	Returns the next 5000 bytes of line. Only complete lines will be returned.
loop over the file object	`for line in f:` `  print line,`	Memory efficient, fast and simple code to loop over each line in the file.
f.tell()	`f.tell()`	Returns the position from the beginning of the file in bytes.
f.seek(offset)	`f.seek(5000)`	Move the file pointer to 5000 bytes from the beginning of the file.
f.seek(offset, from_what)	`f.seek(500, 1)`	from_what modes: 0 is from the beginning of the file, 1 is from the current position, 2 is from the end of the file.
f.write(string)	`f.write('cats and dogs')`	Writes the string to the file.
pickle.dump(x, f)	`pickle.dump(x, f)`	Converts Python object x to a string representation and writes it to the file.
pickle.load(f)	`x = pickle.load(f)`	"Unpickles": returns the Python object from file f.
f.close()	`f.close()`	Close the file to free up system resources.

Team #:_____ Name:_____
Absent:

Write Python code that:

1. Reads the data file P:\810-063-CSIII\temp\phone.txt into a dictionary.
   - Each line of the file contains a name and phone number, e.g., “Fienup 35918”
   - Since the pathname contains the ‘\’ character which is used in a string literal to signal an escape character (e.g., “\a” is the bell, “\t” is the tab, etc.) you’ll need to use a “raw” string literal for the file name as r’P:\810-063-CSIII\temp\phone.txt’ to allow embedded ‘\’ in the string.
2. Allow the user to interactively add new names and phones until some sentinel value (like a <return> at a prompt to enter the next value).
3. save the updated dictionary to a new file (say updatedPhone.txt) in your directory on the P: drive
4. After the program runs, open updatedPhone.txt to see that your changes got saved.