

# Programming Assignment

## Python Statements

1. Write a piece of Python code that prints out the string 'hello world'
2. Write a piece of Python code that prints out the string 'hello world' if the value of an integer variable, happy, is strictly greater than 2.
3. Assume that two variables, varA and varB, are assigned values, either numbers or strings. Write a piece of Python code that evaluates varA and varB, and then prints out one of the following messages:
  - a. "string involved" if either varA or varB are strings
  - b. "bigger" if varA is larger than varB
  - c. "equal" if varA is equal to varB
  - d. "smaller" if varA is smaller than varB
4. Convert the following into code that uses a while loop.  
prints 2  
prints 4  
prints 6  
prints 8  
prints 10  
prints Goodbye !
5. Convert the following into code that uses a while loop.  
prints Hello!  
prints 10  
prints 8  
prints 6  
prints 4  
prints 2
6. Write a while loop that sums the values 1 through end, inclusive. end is a variable that is defined previously. So, for example, if end is defined to be 6, your code should print out the result:  
21  
which is  $1 + 2 + 3 + 4 + 5 + 6$ .
7. Convert the following into code that uses a for loop.  
prints 2  
prints 4  
prints 6  
prints 8  
prints 10  
prints Goodbye !

8. Convert the following into code that uses a for loop.
- ```
prints Hello!  
prints 10  
prints 8  
prints 6  
prints 4  
prints 2
```
9. Write a for loop that sums the values 1 through end, inclusive. end is a variable that is defined previously. So, for example, if end is defined to be 6, your code should print out the result:
- ```
21
```
- which is  $1 + 2 + 3 + 4 + 5 + 6$ .
10. Re-write the code below, but instead of nesting a for loop inside a while loop, nest a while loop inside a for loop.
- ```
iteration = 0  
count = 0  
while iteration < 5:  
    for letter in "hello, world":  
        count += 1  
    print("Iteration " + str(iteration) + "; count is: " + str(count))  
    iteration += 1
```
11. Assume s is a string of lowercase characters.
- Write a program that counts up the number of vowels contained in the string s. Valid vowels are: 'a', 'e', 'i', 'o', and 'u'. For example, if s = 'azcbobobegghakl', your program should print:  
Number of vowels: 5
  - Write a program that prints the number of times the string 'bob' occurs in s. For example, if s = 'azcbobobegghakl', then your program should print  
Number of times bob occurs is: 2
  - Write a program that prints the longest substring of s in which the letters occur in alphabetical order. For example, if s = 'azcbobobegghakl', then your program should print  
Longest substring in alphabetical order is: beggh  
In the case of ties, print the first substring. For example, if s = 'abcbcd', then your program should print  
Longest substring in alphabetical order is: abc

# Python Functions

1. Create a program that guesses a secret number. (the user) thinks of an integer between 0 (inclusive) and 100 (not inclusive). The computer makes guesses, and you give it input - is its guess too high or too low? Using bisection search, the computer will guess the user's secret number. your program should use input to obtain the user's input! Be sure to handle the case when the user's input is not one of h, l, or c. When the user enters something invalid, you should print out a message to the user explaining you did not understand their input. Then, you should re-ask the question, and prompt again for input.
2. Write a Python function, square, that takes in one number and returns the square of that number. This function takes in one number and returns one number.
3. Write a Python function, evalQuadratic(a, b, c, x), that returns the value of the quadratic  $ax^2 + bx + c$ . This function takes in four numbers and returns a single number.
4. Write a Python function, fourthPower, that takes in one number and returns that value raised to the fourth power. You should use the square procedure that you defined in an earlier exercise (you don't need to redefine square in this box; when you call square, the grader will use our definition).
5. Write a Python function, odd, that takes in one number and returns True when the number is odd and False otherwise. You should use the % (mod) operator, not if. This function takes in one number and returns a boolean.
6. Write an iterative function iterPower(base, exp) that calculates the exponential  $\text{base}^{\text{exp}}$  by simply using successive multiplication. For example, iterPower(base, exp) should compute  $\text{base}^{\text{exp}}$  by multiplying base times itself exp times. Write such a function below. This function should take in two values - base can be a float or an integer; exp will be an integer  $\geq 0$ . It should return one numerical value. Your code must be iterative - use of the \*\* operator is not allowed.
7. Write a function recurPower(base, exp) which computes  $\text{base}^{\text{exp}}$  by recursively calling itself to solve a smaller version of the same problem, and then multiplying the result by base to solve the initial problem. This function should take in two values - base can be a float or an integer; exp will be an integer  $\geq 0$ . It should return one numerical value. Your code must be recursive - use of the \*\* operator or looping constructs is not allowed.
8. The greatest common divisor of two positive integers is the largest integer that divides each of them without remainder. For example,  
gcd(2, 12) = 2  
gcd(6, 12) = 6  
gcd(9, 12) = 3  
gcd(17, 12) = 1  
Write an iterative function, gcdIter(a, b), that implements this idea. One easy way to do

this is to begin with a test value equal to the smaller of the two input arguments, and iteratively reduce this test value by 1 until you either reach a case where the test divides both a and b without remainder, or you reach 1.

9. The greatest common divisor of two positive integers is the largest integer that divides each of them without remainder. For example,

$$\text{gcd}(2, 12) = 2$$

$$\text{gcd}(6, 12) = 6$$

$$\text{gcd}(9, 12) = 3$$

$$\text{gcd}(17, 12) = 1$$

A clever mathematical trick (due to Euclid) makes it easy to find greatest common divisors. Suppose that a and b are two positive integers:

If  $b = 0$ , then the answer is a

Otherwise,  $\text{gcd}(a, b)$  is the same as  $\text{gcd}(b, a \% b)$

[See this website for an example of Euclid's algorithm being used to find the gcd.](#)

Write a function `gcdRecur(a, b)` that implements this idea recursively. This function takes in two positive integers and returns one integer.

10. We can use the idea of bisection search to determine if a character is in a string, so long as the string is sorted in alphabetical order.

First, test the middle character of a string against the character you're looking for (the "test character"). If they are the same, we are done - we've found the character we're looking for!

If they're not the same, check if the test character is "smaller" than the middle character. If so, we need only consider the lower half of the string; otherwise, we only consider the upper half of the string. (Note that you can compare characters using Python's `<` function.)

Implement the function `isIn(char, aStr)` which implements the above idea recursively to test if char is in aStr. char will be a single character and aStr will be a string that is in alphabetical order. The function should return a boolean value.

As you design the function, think very carefully about what the base cases should be.

# Tuples, Lists & Dictionaries

1. Write a procedure called `oddTuples`, which takes a tuple as input, and returns a new tuple as output, where every other element of the input tuple is copied, starting with the first one. So if `test` is the tuple `('I', 'am', 'a', 'test', 'tuple')`, then evaluating `oddTuples` on this input would return the tuple `('I', 'a', 'tuple')`.

2. Consider the following sequence of expressions:

```
animals = { 'a': ['aardvark'], 'b': ['baboon'], 'c': ['coati']}
```

```
animals['d'] = ['donkey']
```

```
animals['d'].append('dog')
```

```
animals['d'].append('dingo')
```

We want to write some simple procedures that work on dictionaries to return information.

- a. write a procedure, called `how_many`, which returns the sum of the number of values associated with a dictionary. For example:

```
>>> print(how_many(animals))
```

```
6
```

- b. write a procedure, called `biggest`, which returns the key corresponding to the entry with the largest number of values associated with it. If there is more than one such entry, return any one of the matching keys.

Example usage:

```
>>> biggest(animals)
```

```
'd'
```

If there are no values in the dictionary, `biggest` should return `None`.

3. Suppose we rewrite the `FancyDivide` function to use a helper function.

```
def fancy_divide(list_of_numbers, index):
```

```
    denom = list_of_numbers[index]
```

```
    return [simple_divide(item, denom) for item in list_of_numbers]
```

```
def simple_divide(item, denom):
```

```
    return item / denom
```

This code raises a `ZeroDivisionError` exception for the following call: `fancy_divide([0, 2, 4], 0)`

Your task is to change the definition of `simple_divide` so that the call does not raise an exception. When dividing by 0, `fancy_divide` should return a list with all 0 elements. Any other error cases should still raise exceptions. You should only handle the `ZeroDivisionError`.