DATA304/DATA473/COMP312 - Lab 2

Q1. Define a function add(x,y) that returns the sum, x+y.

In [1]:

```
def add(x,y):
    return x + y

result = add(2,3)
print(result)
```

Q2. Change the definition of add(x,y) so that if y is not given when the function is called, the function just returns x+1. Demonstrate it by adding 2 and 3 (using both arguments) and adding 1 to 2 (using just one).

In [2]:

```
def add(x,y=1):
    return x + y

print(add(2,3))
print(add(2))
```

Q3. Define a function add with two arguments, x and y that returns the sum x+y. The arguments have default values of 3 and 2, respectively. Then call the function with a keyword argument of 5 for y without specifying anything for x and print the result.

In [3]:

```
def add(x=3, y=2):
    return x + y

print(add(y=5))
```

Q4. Change the definition of function add to include a documentation string. Demonstrate it by printing `add.__doc__` .

In [4]:

```
def add(x,y):
    """docstring for add"""
    return x+y

print(add.__doc__)
```

```
docstring for add
```

Q5. The linear congruential generator --a pseudorandom number generator.

In [5]:

```
def linear_congruential(a, c, m, seed, n):
    """linear congruential generator"""
    x = [seed,]
    for j in range(n-1):
        x.append((a*x[j] + c) % m)
    return x

# Example from question
linear_congruential(17, 0, 100, 13, 21)
```

Out[5]:

```
[13,
 21,
 57,
 69,
 73,
 41,
 97,
 49,
 33,
 61,
 37,
 29,
 93,
 81,
 77,
 9,
 53,
 1,
 17,
 89,
 13]
```

In [6]:

```
# Example illustrating full period using conditions given
linear_congruential(5, 3, 256, 1, 257)
```

Out[6]:

```
[1,
 8,
 43,
 218,
 69,
 92,
 207,
 14,
 73,
 112,
 51,
 2,
 13,
 68,
 87,
 182,
 145,
 216,
```

Q6. Gambling on throwin *at least one 6 after four throws of a single fair six-sided die*.

In [7]:

```
import random

random.seed(123)
n = 1000000
count = 0
for k in range(n):
    d1 = random.randint(1,6)
    d2 = random.randint(1,6)
    d3 = random.randint(1,6)
    d4 = random.randint(1,6)
    if ((d1==6) or (d2==6) or (d3==6) or (d4==6)):
        count += 1

print("Results: ", count / n)
```

```
Results:  0.517509
```

Q7. Write a function, words(S), that takes a sentence in a string, S, and returns a list of words, each as a string.

Use your function, words, in a second function, sortedwords(S), that takes a sentence in a string, S and returns a single string holding the words in sorted order, separated by spaces. Hint: refer to the list method sort, and the string method, join.

In [8]:

```
def words(S=''):
    """List of  words in a  sentence"""
    L = S.split()
    return L

def sortedwords(S=''):
    """Sorted words in the sentence"""
    L = words(S)
    L.sort()   # this is done in place
    return ' '.join(L) # separator is the ' '
```

In [9]:

```
assert words()==[],  'words with None wrong'
assert words('now is the  time')==['now', 'is', 'the',  'time'],'words with string wrong'
assert sortedwords()=='','sortedwords of None wrong'
assert sortedwords('now is the  time')=='is now the time','sortedwords with string wrong'
```

Q8. You are given a string containing a series of numbers separated by spaces. For example, the string might be assigned as S = '23.5 34.6 77.9'. Write code to calculate and print the sum of the numbers. Now write Python code in the form of a function string2sum(S). Do not attempt to read the string in, just assign it. Test your function.

In [10]:

```
def string2sum(S):
    """ sum of the numbers in string S """
    numbers = S.split() # list of strings
    total = 0
    for n in numbers:
        total = total + float(n)
    return total

TESTING = True
if TESTING:
    print(string2sum(''))
    print(string2sum('1'))
    print(string2sum('1 1 1 1'))
S = '23.5 34.6 77.9'
print(string2sum(S))
```

```
0
1.0
4.0
136.0
```

Q9. Study different random sampling functions in module `random` of package `numpy` then plot 50 different circles with random coordinates `(x, y)`, random `areas`, and random `colors`, in which `x` and `y` follow an uniform distribution and normal distribution, respectively, while `areas` and `colors` are drawn from two discrete uniform distributions. Hint: study functions `uniform`, `normal`, and `randint`.

In [11]:

```
import math
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

no_of_circles = 50
x = np.random.uniform(-1, 1, no_of_circles)
y = np.random.normal(0, 1, no_of_circles)
areas = [math.pi * np.random.randint(5, 15)**2 for i in range(no_of_circles)]
colors = [np.random.randint(1, 8) for i in range(no_of_circles)]

plt.figure()
plt.scatter(x, y, s=areas, c=colors, alpha=0.8)
plt.show()
```