Data Structures (CS 1520) Lecture 1 Name:____________________

1. The `print` function has optional *keyword arguments* which can be listed last that modify it behavior. The `print` function syntax: `print(value,...,sep=' ',end='\n', file=sys.stdout)`

a) Predict the expected output of each of the following.

Program	**Expected Output**
`print('cat',5,'dog')`	cat␣5␣dog \n
`print()`	\n
`print('cat',5,end='')`	cat␣5␣horse\n
`print(' horse')`	cow\n
`print('cow')`	

Program	**Expected Output**
`print ('cat',5,'dog',end='#',sep='23')`	cat23523dog#
`print ('cat',5,'dog',sep='23','horse')`	error
`print ('cat',5,'dog',sep='>'*3)`	cat>>>5>>>dog \n

'>>>'

2. Review of assignment statements. Predict the output of the following programs

```
a = 123
b = a
a += 1      a = a + 1
print ('a is', a)
print ('b is', b)
```

a → ~~123~~ 124
b → 123

a is 124\n
b is 123\n
int's are immutable

```
c = ['cat', 'dog']
d = c
c.append('cow')
print('c is', c)
print('d is', d)
```

c, d → [cat, dog, cow]

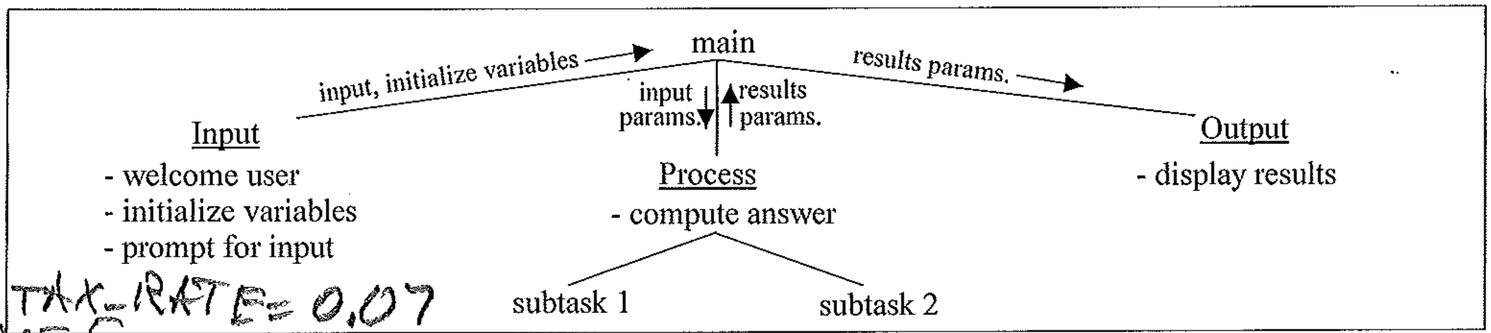
c is ['cat', 'dog', 'cow']
d is ['cat', 'dog', 'cow']
lists are mutable

```
c = 'cat'
d = c
c += 'fish'     c = c + 'fish'
print('c is', c)
print('d is', d)
```

c → catfish
d → cat

c is catfish
d is cat
strings are immutable

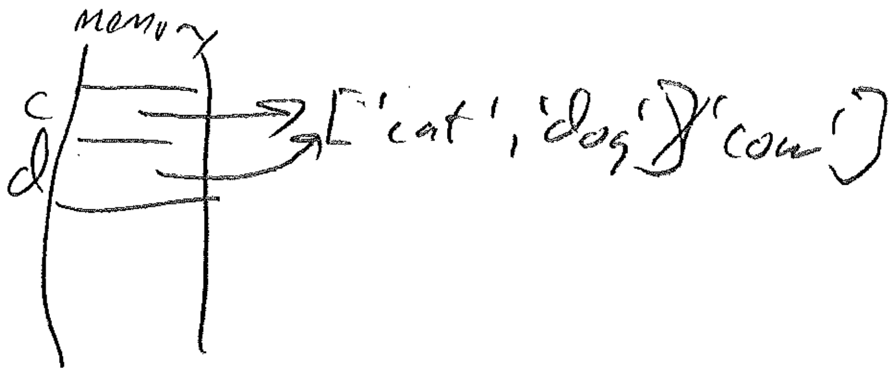
Most simple programs have a similar functional-decomposition design pattern (IPO - Input, Process, Output):

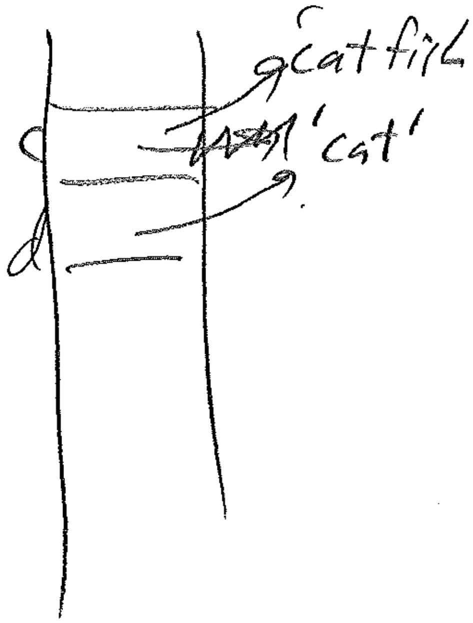


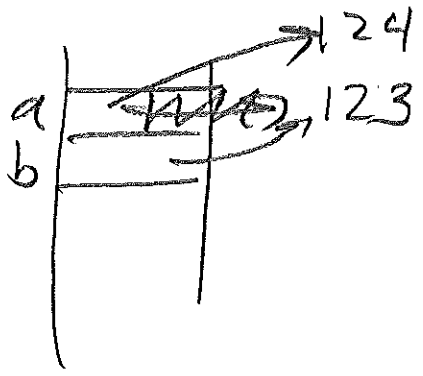
TAX_RATE = 0.07
x = 5

```
""" Simple IPO program to sum a list of numbers. """
def main():
    label, values = getInput()      (**)
    total = sum(values)
    displayResults(label, total)

def getInput():
    """ Get label and list of values to sum."""
    label = input("What are we summing? ")
    numberOfValues = int(input("How many values are there? "))
    values = []      list()
    for i in range(numberOfValues):
        values.append(eval(input("Enter the next number: ")))
    return label, values

def displayResults(label, total):
    """ Display sum of values. """
    print("The sum of", label, "values is", total)
                                                    None
main()   # starts the main function running
(*)
```

```
What are we summing? money
How many values are there? 4
Enter the next number: 10
Enter the next number: 20
Enter the next number: 30
Enter the next number: 50
The sum of money values is 110
```

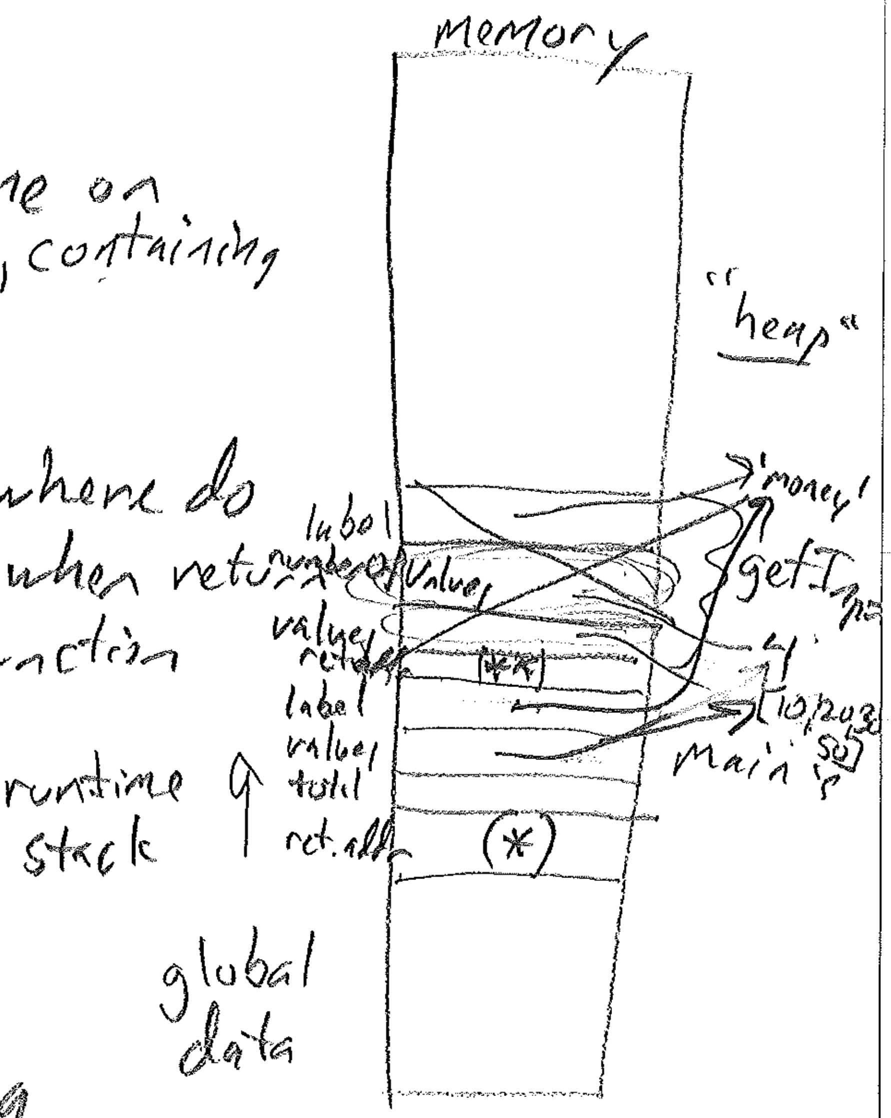

memory
c
d
['cat', 'dog'] ['cow']


'catfish'
c
'cat'
d


124
a
123
b

Function Calls

on a call – push a call-frame on run-time stack, containing
(1) local variables
(2) parameters
(3) return addr. – where do I go back to when return or end of function



on return

Call-frame popped from run-time stack with execution continuing at return address.

Return values assigned at return addr.
(Note: Default value None returned if nothing explicitly returned.)

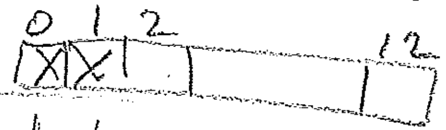
3. Design a program to roll two 6-sided dice 1,000 times to determine the percentage of each outcome (i.e., sum of both dice). Report the outcome(s) with the highest percentage.

a) Customize the diagram for the dice problem by briefly describing what each function does and what parameters are passed.

import random

~~random~~ random.randint(1,6)

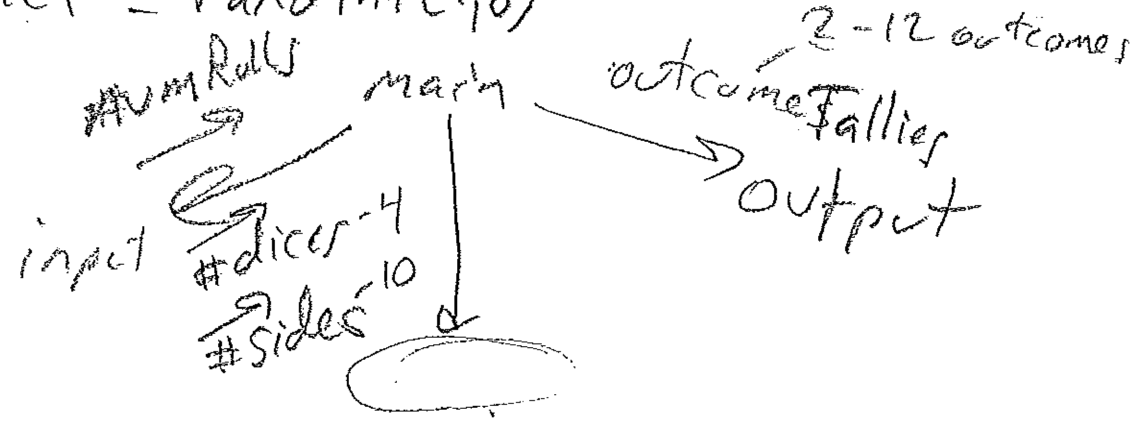
List outcomeTallies



from random import randint

die1 = randint(1,6)

dictionary
{2:15, ..



b) An alternative design methodology is to use object-oriented design. For the above dice problem, what objects would be useful and what methods (operations on the objects) should each perform?