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

Program	Expected Output
`print ('cat',5,'dog',end='#',sep='23')`	cat23523dog#
`print ('cat',5,'dog',sep='23','horse')`	~~cat23523~~ 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
print ('a is', a)
print ('b is', b)
```

a is 124
b is 123

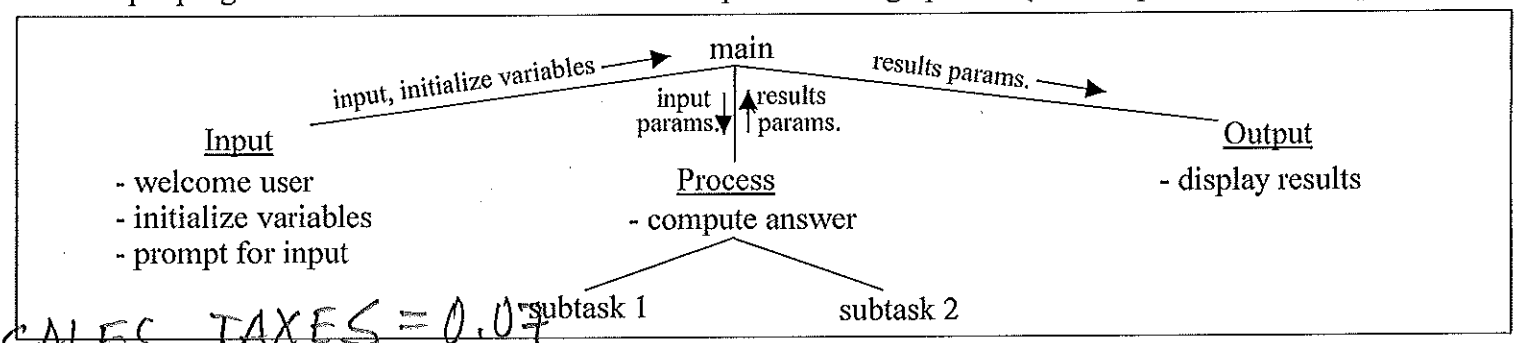
```
c = ['cat', 'dog']
d = c
c.append('cow')
print('c is', c)
print('d is', d)
```

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

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

c is catfish
d is cat

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

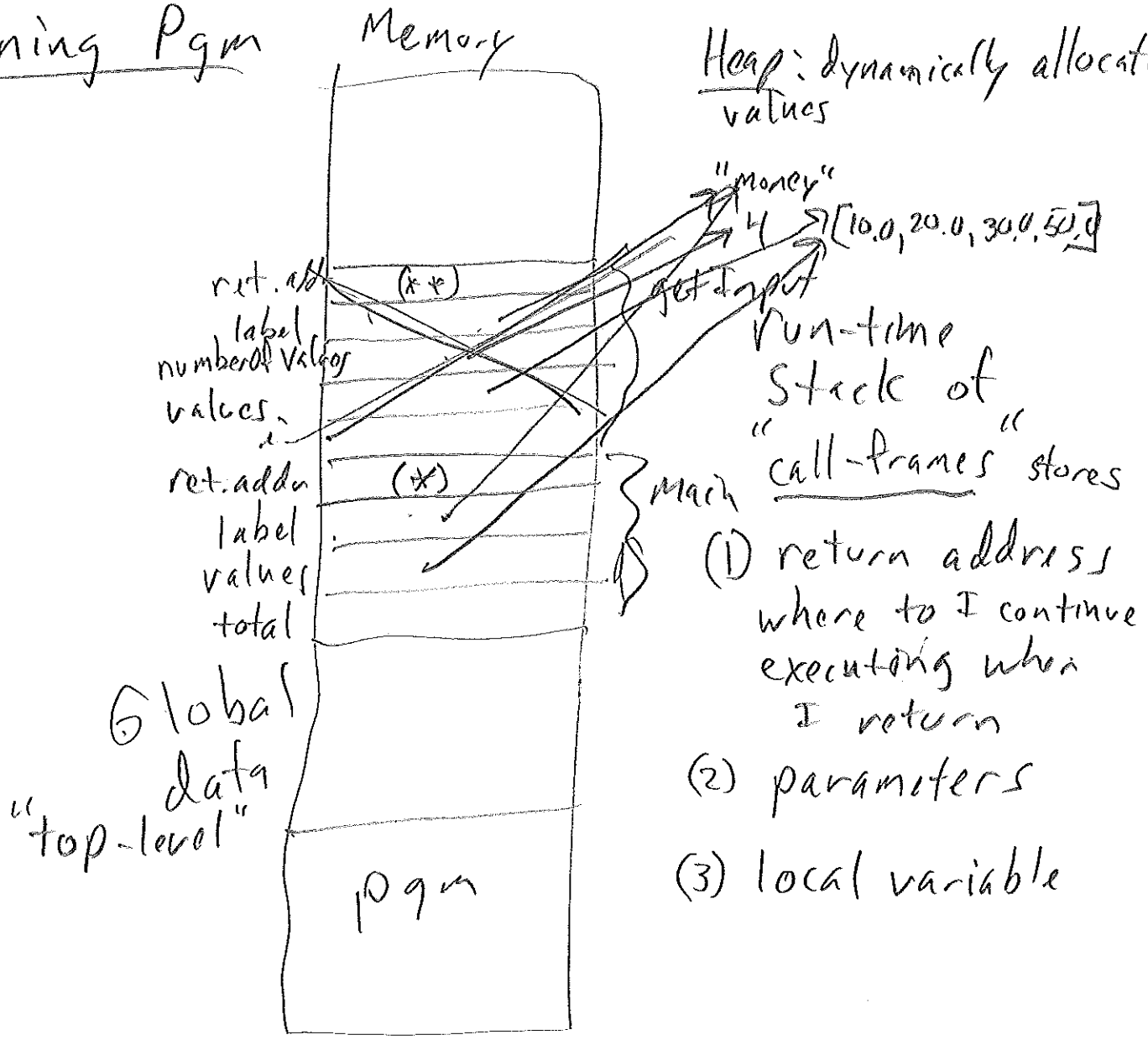


SALES_TAXES = 0.07

```
""" 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 = []
    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)

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
```

Running Pgm

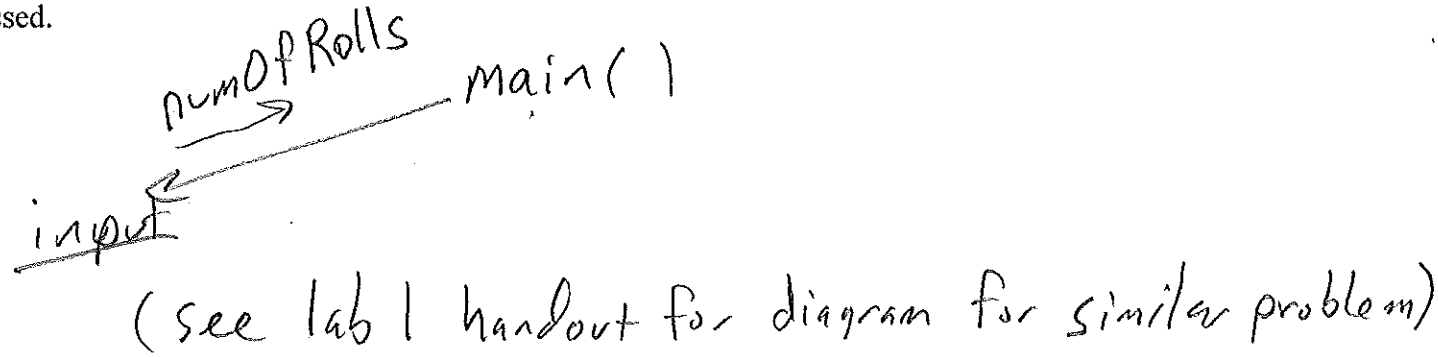

Memory
Heap: dynamically allocated values
"money"
[10.0, 20.0, 30.0, 50.0]
ret. add
label
number of values
values
(**)
get_input
ret. addr
label
values
total
(*)
Global data
"top-level"
Pgm
Run-time Stack of "call-frames" stores
each
(1) return address where to I continue executing when I return
(2) parameters
(3) local variable

outcome[] 0 1 2 3 4 5 6 7 8 9 10 11 12

randint(1,6)

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. import random

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



(see lab 1 handout for diagram for similar problem)

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?

Data Structures (CS 1520) Lab 1 Name:_____

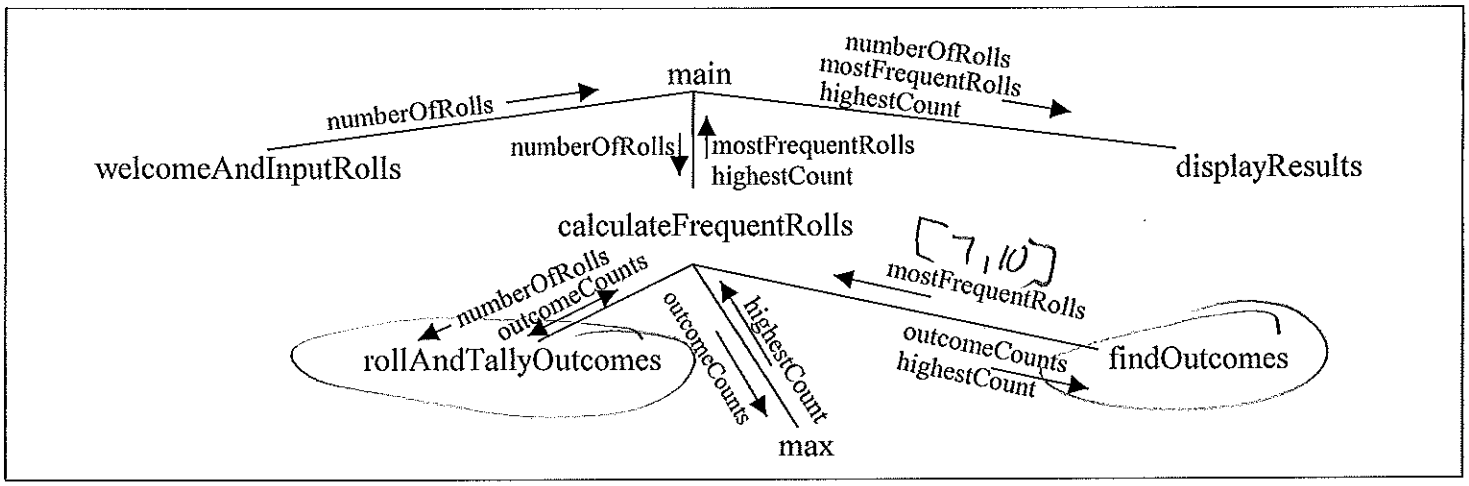
Objective: To practice writing Python code.

To start the lab: Download and unzip the file lab1.zip from
http://www.cs.uni.edu/~fienup/cs1520f18/labs/lab1.zip

Part A: In the folder lab1, open the diceOutcomes.py program in IDLE. (Right-click on diceOutcomes.py | Edit with IDLE) It contains a partial program we started to discuss in class to solve the problem:

> "Write 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."

I decided to functional-decomposition this problem as:



`main` - provides an outline of program by calling top-level functions
`welcomeAndInputRolls` - Displays welcome message for the user. Gets and returns the number of dice rolls from the user.
`calculateFrequentRolls` - Rolls the dice the correct number of times, tallies the outcomes, and returns a list of outcomes with the highest count and highest count.
`rollAndTallyOutcomes` - Rolls the dice the correct number of times and tallies the outcomes. Returns a list of tallies with the index being the outcome.
`max` - built-in function to return the largest item in an iterable data structure like a list.
`findOutcomes` - Returns a list of outcomes with the highest count.
`displayResults` - Displays the outcome(s) with the highest percentage.

Consider running the program with only 10 dice rolls instead of 1,000. The program output with some extra debugging prints showing the two Python lists used: outcomeCounts and mostFrequentRolls.

```
This programs rolls two 6-sided dice many times to
determine the outcome(s) with the highest percentage.
How many times would you like to roll the pair of dice? 10
outcomeCounts: [0, 0, 1, 0, 2, 1, 0, 3, 0, 0, 3, 0, 0]
mostFrequentRolls: [7, 10] and highestCount: 3
The highest percentage is 30.0 for outcome(s): 7 10
```

Your task for lab 1 is to complete the code for the `rollAndTallyOutcomes` and `findOutcomes` functions.

After you have working code, raise your hand and demonstrate your code.

If you complete all parts of the lab, nothing needs to be turned in for this lab. If you do not get done today, then show me the completed lab in next week's lab period. When done, remember save your program to a USB drive, email to yourself, Google drive, etc.