HW 3 References

(3 pts) question 1 tests stepping through code and edge-cases (section 2.2.1)
(2 pts) question 2 connects arrays back to the ISA. (section 3.3 and lecture 5)
(1 pt) question 3 tests logical operators and modulus (section 3.4 and 2.3.2)
(2 pts) question 4 tests logical operators and modulus (section 3.4 and 2.3.2)
(2 pts) question 5 tests functions and scoping (section 3.5 and 3.5.2)
(5 pts) question 6 tests if-else, debugging, edge cases, and logic errors (section 1.4.3.3, 2.2.1, 2.4.4.1)
(3 pts) question 7 tests Python syntax and syntax errors (section 1.4.3.3 and 2.4.1)
(7 pts) question 8 tests loops (section 2.4.4.3)
(3 pts) question 9 tests arrays (section 3.3)
(3 pts) question 10 tests functions (section 3.5)

HW 3 Solutions

1. Tests edge-cases (section 2.2.1)

```
          m = 3
          n = 0
          product = m
     (i)  while(n > 1):
     (ii)       product = product + m
     (iii)      n = n - 1
     (iv) print(product)
```

a.

Current line	**Calculation performed by this line**	**Variables**			**Next line:**
		`m`	**`n`**	**`product`**	
i	checking in n > 1	3	0	3	iv
iv	printing product	3	0	3	end

b.
The code does not work for edge case (or corner case) n == 0.
Off-by-one error is also be accepted.

Fix:

```
m = 3
n = 0
product = 0                    # initialized the product to 0
while(n > 0):                  # continue while n > 0
      product = product + m
      n = n - 1
print(product)
```

2. Tests arrays and connects it back to the ISA (section 3.3 and lecture 5)

Sample solution 1:

Yes, I can write an ISA instruction for accessing the 9th character of the 5th string. My ISA allocates a fixed length of 256 characters for each string. Therefore, if the array is in r1, I can do

ld r1, 4, 8

r1 because the array location is stored in r1
4 because it is the 5th string (offset from 0)
8 because it is the 9th character (offset from 0)

Sample solution 2:

No. It is difficult because strings have a variable length. Therefore, there is no immediate values that I can pass to access such a character.

3. Tests logical operators and modulus (section 3.4 and 2.3.2)

`0, 1, 2, 3, 10`

4. Tests logical operators and modulus (section 3.4 and 2.3.2)

a. 1 will satisfy the condition
b. 3 fails to satisfy `(x%2==0 or x%3==1)`
c. 4 fails to satisfy `(x!=4)`
d. 10 will satisfy the condition

5. Tests functions and scoping (section 3.5 and 3.5.2)

```
def bar(y):
   print(y)
   x = 5
   return x
y = 3
print(bar(y))
print(y)
```

3
5
3

6. Tests if-else, debugging, edge cases, and logic errors (section 1.4.3.3, 2.2.1, and 2.4.4.1)

a.
Method 1:

```
if(number > 0):
      print("positive")
else:
      if(numbe r <= 0):
            print("not positive")
```

b.
Method 2:

```
results = ""
if(number <= 0):
      results = "not "
results = results  + "positive"
print(results)
```

c.
Method 3 worked as is:

```
if(number > 0):
      print("positive")
else:
      print("not positive")
```

7. Tests Python syntax and syntax errors (section 1.4.3.3 and 2.4.1)

```
h3110W0RlD = "Hello World"
_ = "h3110W0RlD"
weAreTheChampions = 1
# infinite loop to fight 'til the end
while(weAreTheChampion s == 1):   # 2 syntax errors on this line
                                  # should have use == for comparison
                                  # missing colon at end of while
                                  # note: Champions was misspelled
     print("f1gh71ng \'til the end")
fahrenhype = -40            # int is not valid Python syntax
celsius = (9 / 5) * (fahrenhype + 32) # note:missing close parenthesis
print(celsius)                    # note: celsius had wrong case
```

8. Tests loops (section 2.4.4.3)

```
import input
positiveOddNumberInput = input.get_num("Enter a number: ")

print("begin printing horizontal line")
lineToPrint = ""
i = 0
while(i < positiveOddNumberInput):
   lineToPrint = lineToPrint + "*"
   i = i + 1
print(lineToPrint)
print("end printing horizontal line")

print("begin printing vertical line")
i = 0
while(i < positiveOddNumberInput):
   print("*")
   i = i + 1
print("end printing vertical line")
```

```
print("begin printing triangle 1")
lineToPrint = ""
i = 0
while(i < positiveOddNumberInput):
   lineToPrint = lineToPrint + "*"
   print(lineToPrint)
   i = i + 1
print("end printing triangle 1")

print("begin printing triangle 2")
i = 0
while(i < 1 + (positiveOddNumberInput / 2)):
   whitespaceToPrint = ""
   j = i
   while(j < positiveOddNumberInput / 2):
      whitespaceToPrint = whitespaceToPrint + " "
      j = j + 1
   asterisksToPrint = "*"
   j = 0
   while(j < i):
      asterisksToPrint = asterisksToPrint + "**"
      j = j + 1
   lineToPrint = whitespaceToPrint + asterisksToPrint
   print(lineToPrint)
   i = i + 1
print("end printing triangle 2")
```

9. Tests arrays (section 3.3)

```
n = 15
i = 2
fa = [0,1]
while(i < n):
   fa = fa + [fa[i-1] + fa[i-2]]
   i = i + 1
print(fa)
```

10. Tests functions (section 3.5)

```
def ftoc(F):
   return (F - 32) * 5 / 9
F = -50
while(F <= 50):
   x = 4
   C = ftoc(F)
   if(F == C):
      print("Fahrenheit and Celsius are equal at -40 degrees!")
   else:
      print("F=" + F + ", " + "C=" + C)
   F = F + 1
```