

## Multiple choice questions:

1. What is the output of the following program?

```
def f(x, y):  
    if y == 0:  
        return x  
    else:  
        return f(y, x%y)
```

```
print(f(240, 150))
```

- a. 30
  - b. 60
  - c. 90
  - d. 120
  - e. This program will produce no output
2. Given the class definition for 'Foo' which of the following is the correct way to access attribute 'num' inside 'fun'?

```
class Foo:  
    # Assume an attribute called 'num' has been created & set  
    def fun(self):
```

- a. num
  - b. fun.num
  - c. num.num
  - d. self.num
  - e. None of the above
3. Given the previous class definition which of the following is the correct way to display attribute 'num' inside 'main'?

```
def main():  
    aFoo = Foo()
```

- a. print(num)
- b. print(aFoo.num)
- c. print(num.num)
- d. print(main.num)
- e. None of the above

4. Based on what you see in the program below which of the following is true of 'num'?

```
num = 1
def fun1(num):
    print(num)
```

- a) num is a local variable.
- b) num is a global variable.
- c) num is a parameter to the function.
- d) There is a syntax error in the creation of num.
- e) There are two instances of num, one is global and the other is a function parameter (a type of local).

For Questions 5 – 6 you are to specify the output of the corresponding program.

5.

```
aString1 = "snoopy"
aString2 = "cartman"
print(aString1[2]+aString2[4:])
```

- a. oman
- b. ting2
- c. ntman
- d. ortman
- e. snoopycartman

6.

```
x = ['!',4.0]
y = x
x[1] = 'b'
print(x,y)
```

- a) x y
- b) ! b ! b
- c) ['!', 'b'] ['!', 4.0]
- d) ['!', 4.0] ['!', 'b']
- e) ['!', 'b'] ['!', 'b']
- f)

7. Based on the program you see below how many 'Person' objects have been created?

```
class Person:
    def __init__(self):
        age = 0

    def setAge(self,newAge):
        self.age = newAge
```

- a. 0
  - b. 1
  - c. 2
  - d. 3
  - e. >3
8. Assume that class 'Person' has already been defined. How many 'Person' objects have been created?

```
a = Person()
b = a
d = b
a.setAge(888)
```

- a. 0
- b. 1
- c. 3
- d. 4
- e. 888

**Written questions:**

For the short answer questions when specifying program output, whenever it applies, if at all) you can specify a space with a `<SP>` and a tab with a `<tab>`. A newline can be specified by moving the output that follows to the next line.

**Short answer 1:**

1. Write two functions that perform file output.
  - a. `file1(num)` takes an integer as a parameter. It will write to a file the values 1 to that integer (inclusive) to a file each number on a separate line.
  - b. `file2(num)` takes an integer as a parameter. It will write to a file the values 0 to that integer (exclusive) to a file with all numbers on the same line and numbers will be separated by a comma.

**Short answer 2:**

What is the output of the following program?

```
# Program: file_trace.py
f = open("data.txt", "r")
i = 1
for l in f:
    if ((i % 2) == 0):
        a = l.split(',')
    else:
        a = l.split()
    print(a)
    i = i + 1
f.close()
```

**# Input file: input.txt**

Blue sky to forever

The green grass blows in the wind, dancing

It would be a much better sight with you, with me

If you hadn't met me, I'd be fine on my own, baby

Never felt so lonely, then you came along

- Akira Yamaoka (Silent Hill 3 lyrics)

<< Output >>

**Short answer 3:**

What is the output of the following program (assume that there is only one file in the same directory as the program which is called "input.txt" and the user doesn't enter a path) if the user enters: (a) input.txt (b) dat.txt

**# Program**

```
ok = False
```

```
while (ok == False):
```

```
    try:
```

```
        fn = input("Name of input file: ")
```

```
        fv = open(fn,"r")
```

```
    except IOError:
```

```
        print("Can't open ", fn)
```

```
    else:
```

```
        print("Opening ", fn)
```

```
for l in fv:
```

```
    print("%s\t" %(l))
```

```
fv.close()
```

**# Input.txt**

```
Nous sommes des dégourdis,
```

```
Nous sommes des lascars
```

```
Des types pas ordinaires.
```

```
Nous avons souvent notre cafard,
```

```
Nous sommes des légionnaires.
```

```
Au Tonkin, la Légion immortelle
```

```
À Tuyen-Quang illustra notre drapeau,
```

```
Héros de Camerone et frères modèles
```

```
- Common marching cadence (French foreign legion)
```

<< Program output >>

**Short answer 4:**

Write a function called `findHighest()` that will find and return the highest grade in the list called `grades` in the following program. You can assume that the list has already been properly initialized (and not empty). The number of elements in the list is determined by the constant called `SIZE`.

```
SIZE = 10
```

```
# Write your answer here
```

```
# End of answer space
```

```
def main ():  
    grades = []  
    grades = initialize()  
    display(grades)  
    highest = findHighest(grades)  
    print(highest)
```

```
main ()
```

**Short answer 5:**

Write the output of the following program in the space provided.

```
a = 1
b = 2

def fun():
    a = 10
    b = 20

def fun1(a,b,c,d,e):
    a = a + a
    b = b + b
    c = c + c
    d = ['x','y','z']
    e = "bar"

def fun2(d,e):
    d[0] = "X"
    d[1] = "Y"
    d[2] = "Z"
    e = e + "c"
    return(e)

def start():
    b = 3
    c = 4
    d = [10,'a',"foo"]
    e = "cps"
    print(a, b, c, d, e)
    fun()
    print(a, b)
    fun1(a,b,c,d,e)
    print(a, b, c, d, e)
    fun2(d,e)
    print(d, e)

start()
```

**# Answer:**

If you wish you can use the `len()` function for the next 2 questions. `len()` takes a string or list as a parameter and returns its length (number of elements).

**Short answer 6:**

For the following program write a function called `fileWrite()` that will take the list `'grades'` as a parameter. It will write the grades to an output file called `'data.txt'`. Each grade will reside on its own line.

```
def initialize():  
    # Assume it creates the list properly and returns it back  
    # to the caller.  
    return(grades)
```

**# Write your answer here**

```
def start():  
    grades = initialize()  
    fileWrite(grades)  
  
start()
```

**Short answer 7:**

Write a function called `isNatural()` that will take the string `aString` as a parameter. It will return `True` if the string is a natural number (non-negative integer) and `False` otherwise. You are to write the checking code yourself rather than calling a pre-created function.

**# Begin function definition**

**# End answer space**

```
def start():
    aString = input("Enter an integer value: ")
    if (isInteger(aString) == True):
        print("Integer")
    else:
        print("Not integer")
start()
```

## Short answer 8 (eventually an answer will be posted, try on your own first)

Conway's Game of Life is a simple but well-known biological simulation.

(Exert from Math.com)

Life is played on a grid of square cells--like a chess board but extending infinitely in every direction. A cell can be *live* or *dead*. A live cell is shown by putting a marker on its square. A dead cell is shown by leaving the square empty. Each cell in the grid has a neighborhood consisting of the eight cells in every direction including diagonals.

To apply one step of the rules, we count the number of live neighbors for each cell. What happens next depends on this number.

- A dead cell with exactly three live neighbors becomes a live cell (birth).



- A live cell with two or three live neighbors stays alive (survival).



- In all other cases, a cell dies or remains dead (overcrowding or loneliness).



Write a function called "count()" that will take three parameters:

- a 2D list (the 'biosphere' that contains the pattern of life forms)
- the location being scanned (row/column coordinates)

For each of the 'inner squares' (row = 1 to row = 6 / column = 1 to column = 8) in the example program below this function will be used to count the number of neighbors in the 8 surrounding squares. The function will then return this value back to the caller. You can assume that other tasks such as scanning the outer squares (e.g., all elements on row zero) or the actual 'births' or 'deaths' of life forms is handled elsewhere in the program.

```

MAX_ROWS = 8
MAX_COLUMNS = 8
LIFE = "*"
EMPTY = " "
MUTANT = "!"
MIN_NEIGHBORS = 2
MAX_NEIGHBORS = 3
OPTIMAL_NEIGHBORS = 3
MAX_TURNS = 10

```

**# Included for reference only**

```

def display(biosphere):
    r = 0
    while (r < MAX_ROWS):
        c = 0
        while (c < MAX_COLUMNS):
            print(biosphere[r][c], end = "")
            c = c + 1
        print()
        r = r + 1

```

**# Included so you can see how your function will be used.**

```

def checkInnerSquares(biosphere):
    r = 1
    while (r < (MAX_ROWS-1)):
        c = 1
        while (c < (MAX_COLUMNS-1)):
            numNeighbors = count(biosphere,r,c)
            if (biosphere[r][c] == LIFE):
                if (numNeighbors < MIN_NEIGHBORS):
                    biosphere[r][c] = EMPTY
                elif (numNeighbors > MIN_NEIGHBORS):
                    biosphere[r][c] = EMPTY
            elif (biosphere[r][c] == EMPTY):
                if (numNeighbors == OPTIMAL_NEIGHBORS):
                    biosphere[r][c] = LIFE
            c = c + 1
        r = r + 1

```

```
def count(biosphere,row,column):  
    # Write your solution here
```

```
def determineBirthsDeaths(biosphere):  
    # Assume that the other squares are checked by other  
    # functions  
    checkInnerSquares(biosphere)
```

**# Randomly generates a population for the whole biosphere. You # can assume that it works properly, no need to trace the # code.**

```
def initialize():
    biosphere = []
    r = 0
    while (r < MAX_ROWS):
        c = 0
        biosphere.append([])
        while (c < MAX_COLUMNS):
            biosphere[r].append(populateLocation())
            c = c + 1
        r = r + 1
    return(biosphere)
```

**# Populates a single location in the biosphere. You can assume # that it properly works, no need to trace the code.**

```
def populateLocation():
    temp = random.randint(1,100)
    if (temp >= 1) and (temp <= 25):
        location = LIFE
    elif (temp > 25) and (temp <= 100):
        location = EMPTY
    else:
        location = MUTANT
    return(location)
```

```
def start():
    biosphere = initialize()
    for currentTurn in range (0,MAX_TURNS,1):
        display(biosphere)
        determineBirthsDeaths(biosphere)
        input("Hit enter to continue")
start()
```

(Additional resources related to this question “Game of Life”, not mandatory to write the function but may be of interest if you want further details about the ‘game’ or you wish to see sample implementations)

- Math.com rules: <http://www.math.com/students/wonders/life/life.html>
- Java animated execution: <http://www.bitstorm.org/gameoflife/>
- Python solution in one line (correctness not verified by the instructor): <http://www.petercollingridge.co.uk/blog/python-game-of-life-in-one-line>

**Short answer 9:**

For this question refer to the definition of class “Ninja” and the “start()” function.

```
class Ninja:
    lives = -1
    clan = ""
    specialPower = ""

def start():
    sho = Ninja(9, "Kuji-kuri", "Invisibility")

    # Make a copy
    cho = sho.clone()

    # This only changes the attributes of the copy, the
    # original is unchanged. (Hint: it's not a ninja-magic
    # special ability; just take advantage of the fact that
    # the constructor returns a reference to an object).
    cho.clan = "Kanakuna"
    cho.power = "Yogen"

start()
```

**Part (a):**

Write an ‘init()’ constructor that will take 3 parameters when called. The method will set the ‘lives’, ‘clan’ and ‘power’ attributes to the values of these three parameters.

**Part (b):**

Write a ‘clone()’ method. This method will create a copy of the ninja whose clone method has been called (same values for all attributes but separate object is created and returned by this method). In the ‘start()’ function ‘sho’ and ‘cho’ refer to two separate Ninja objects. When the cho object is modified, the attributes of sho object retain their original values.

**Short answer 10:**

Describe what does function 'modify()' do in the program below. (How does 'newString' differ from the original 'oldString')?

```
def inLowRange(aChar):
    aNum = ord(aChar)
    if (aNum >= 97) and (aNum <= 122):
        return(True)
    else:
        return(False)

def inUpRange(aChar):
    aNum = ord(aChar)
    if (aNum >= 65) and (aNum <= 90):
        return(True)
    else:
        return(False)

def modify(oldString):
    size = len(oldString)
    i = 0
    newString = ""
    while (i < size):
        if (inLowRange(oldString[i]) == True):
            newString = newString +
                chr((ord(oldString[i]) - 32))
        elif (inUpRange(oldString[i]) == True):
            newString = newString +
                chr((ord(oldString[i]) + 32))
        else:
            newString = newString + oldString[i]
        i = i + 1
    return(newString)

def start():
    oldString = input("Enter a series of characters: ")
    newString = modify(oldString)
    print(oldString, newString)
```

For your reference the table of the basic ASCII values is provided on the next page (Look at the column labelled 'Dec')

| Char  | Dec | Oct  | Hex  | Char | Dec | Oct  | Hex  | Char | Dec | Oct  | Hex  | Char  | Dec | Oct  | Hex  |
|-------|-----|------|------|------|-----|------|------|------|-----|------|------|-------|-----|------|------|
| (nul) | 0   | 0000 | 0x00 | (sp) | 32  | 0040 | 0x20 | @    | 64  | 0100 | 0x40 | `     | 96  | 0140 | 0x60 |
| (soh) | 1   | 0001 | 0x01 | !    | 33  | 0041 | 0x21 | A    | 65  | 0101 | 0x41 | a     | 97  | 0141 | 0x61 |
| (stx) | 2   | 0002 | 0x02 | "    | 34  | 0042 | 0x22 | B    | 66  | 0102 | 0x42 | b     | 98  | 0142 | 0x62 |
| (etx) | 3   | 0003 | 0x03 | #    | 35  | 0043 | 0x23 | C    | 67  | 0103 | 0x43 | c     | 99  | 0143 | 0x63 |
| (eot) | 4   | 0004 | 0x04 | \$   | 36  | 0044 | 0x24 | D    | 68  | 0104 | 0x44 | d     | 100 | 0144 | 0x64 |
| (enq) | 5   | 0005 | 0x05 | %    | 37  | 0045 | 0x25 | E    | 69  | 0105 | 0x45 | e     | 101 | 0145 | 0x65 |
| (ack) | 6   | 0006 | 0x06 | &    | 38  | 0046 | 0x26 | F    | 70  | 0106 | 0x46 | f     | 102 | 0146 | 0x66 |
| (bel) | 7   | 0007 | 0x07 | '    | 39  | 0047 | 0x27 | G    | 71  | 0107 | 0x47 | g     | 103 | 0147 | 0x67 |
| (bs)  | 8   | 0010 | 0x08 | (    | 40  | 0050 | 0x28 | H    | 72  | 0110 | 0x48 | h     | 104 | 0150 | 0x68 |
| (ht)  | 9   | 0011 | 0x09 | )    | 41  | 0051 | 0x29 | I    | 73  | 0111 | 0x49 | i     | 105 | 0151 | 0x69 |
| (nl)  | 10  | 0012 | 0x0a | *    | 42  | 0052 | 0x2a | J    | 74  | 0112 | 0x4a | j     | 106 | 0152 | 0x6a |
| (vt)  | 11  | 0013 | 0x0b | +    | 43  | 0053 | 0x2b | K    | 75  | 0113 | 0x4b | k     | 107 | 0153 | 0x6b |
| (np)  | 12  | 0014 | 0x0c | ,    | 44  | 0054 | 0x2c | L    | 76  | 0114 | 0x4c | l     | 108 | 0154 | 0x6c |
| (cr)  | 13  | 0015 | 0x0d | -    | 45  | 0055 | 0x2d | M    | 77  | 0115 | 0x4d | m     | 109 | 0155 | 0x6d |
| (so)  | 14  | 0016 | 0x0e | .    | 46  | 0056 | 0x2e | N    | 78  | 0116 | 0x4e | n     | 110 | 0156 | 0x6e |
| (si)  | 15  | 0017 | 0x0f | /    | 47  | 0057 | 0x2f | O    | 79  | 0117 | 0x4f | o     | 111 | 0157 | 0x6f |
| (dle) | 16  | 0020 | 0x10 | 0    | 48  | 0060 | 0x30 | P    | 80  | 0120 | 0x50 | p     | 112 | 0160 | 0x70 |
| (dc1) | 17  | 0021 | 0x11 | 1    | 49  | 0061 | 0x31 | Q    | 81  | 0121 | 0x51 | q     | 113 | 0161 | 0x71 |
| (dc2) | 18  | 0022 | 0x12 | 2    | 50  | 0062 | 0x32 | R    | 82  | 0122 | 0x52 | r     | 114 | 0162 | 0x72 |
| (dc3) | 19  | 0023 | 0x13 | 3    | 51  | 0063 | 0x33 | S    | 83  | 0123 | 0x53 | s     | 115 | 0163 | 0x73 |
| (dc4) | 20  | 0024 | 0x14 | 4    | 52  | 0064 | 0x34 | T    | 84  | 0124 | 0x54 | t     | 116 | 0164 | 0x74 |
| (nak) | 21  | 0025 | 0x15 | 5    | 53  | 0065 | 0x35 | U    | 85  | 0125 | 0x55 | u     | 117 | 0165 | 0x75 |
| (syn) | 22  | 0026 | 0x16 | 6    | 54  | 0066 | 0x36 | V    | 86  | 0126 | 0x56 | v     | 118 | 0166 | 0x76 |
| (etb) | 23  | 0027 | 0x17 | 7    | 55  | 0067 | 0x37 | W    | 87  | 0127 | 0x57 | w     | 119 | 0167 | 0x77 |
| (can) | 24  | 0030 | 0x18 | 8    | 56  | 0070 | 0x38 | X    | 88  | 0130 | 0x58 | x     | 120 | 0170 | 0x78 |
| (em)  | 25  | 0031 | 0x19 | 9    | 57  | 0071 | 0x39 | Y    | 89  | 0131 | 0x59 | y     | 121 | 0171 | 0x79 |
| (sub) | 26  | 0032 | 0x1a | :    | 58  | 0072 | 0x3a | Z    | 90  | 0132 | 0x5a | z     | 122 | 0172 | 0x7a |
| (esc) | 27  | 0033 | 0x1b | ;    | 59  | 0073 | 0x3b | [    | 91  | 0133 | 0x5b | {     | 123 | 0173 | 0x7b |
| (fs)  | 28  | 0034 | 0x1c | <    | 60  | 0074 | 0x3c | \    | 92  | 0134 | 0x5c |       | 124 | 0174 | 0x7c |
| (gs)  | 29  | 0035 | 0x1d | =    | 61  | 0075 | 0x3d | ]    | 93  | 0135 | 0x5d | }     | 125 | 0175 | 0x7d |
| (rs)  | 30  | 0036 | 0x1e | >    | 62  | 0076 | 0x3e | ^    | 94  | 0136 | 0x5e | ~     | 126 | 0176 | 0x7e |
| (us)  | 31  | 0037 | 0x1f | ?    | 63  | 0077 | 0x3f | _    | 95  | 0137 | 0x5f | (del) | 127 | 0177 | 0x7f |

&lt;&lt; Write your answer here &gt;&gt;

**Short answer 11:**

Modify the following program so that each column is numbered from 0 – 9 (before the list is displayed). Also prior to displaying each row each row is numbered from 0 – 9. The code has been modified in the output below so that each element is not randomly initialized but instead each element is set to zero.

```

0123456789
0 0000000000
1 0000000000
2 0000000000
3 0000000000
4 0000000000
5 0000000000
6 0000000000
7 0000000000
8 0000000000
9 0000000000

```

```

import random
aGrid = []
SIZE = 10

# Creating the grid data
for r in range (0,SIZE,1):
    aGrid.append ([])
    for c in range (0,SIZE,1):
        element = random.randint(0,9)
        aGrid[r].append (0)

# Displaying the grid
for r in range (0,SIZE,1):
    for c in range (0,SIZE,1):
        print(aGrid[r][c], end="")
    print()

```