

## NAME: ANSWER KEY

The rules:

- Relax!
- For reference you may use the summary sheets provided and two sheets of your own notes (you may write on the front and back of your sheets).
- All work must be your own. Merely *looking* at the work of others is cheating and may carry all the consequences associated with cheating.
- Generally the interactive prompt will not be shown even when the code is assumed to have been entered interactively.
- Neatness counts. If I can't easily read it, you won't get credit. Indicate indentation clearly where indentation is necessary.
- For questions whose answer is either a `float` or an `int`, you must indicate the type of the answer by either including or not including a decimal point.
- You may use the "standard" scientific functions of a calculator. Programmed calculations are not permitted. (A programmable calculator is allowed, but you are not allowed to program it or use pre-installed programs.)
- This is not a race! **If you submit your test before 50 minutes from the start of the exam, the number of incorrect points will be multiplied by three.** Take your time and check your work!
- The value of each question is indicated within brackets, e.g., [10]. (Questions with equal value are not necessarily of equal difficulty.)

1. [4] What output is produced by the following code?

```
xlist = [1, [2, 3], 4]
print(xlist[2])
```

- (a) 2
- (b) [2]
- (c) [2, 3]
- (d) 4  $\Leftarrow$  **ANSWER**
- (e) [4]

2. [4] What output is produced by the following code?

```
xlist = [1, [2, 3], 4]
print(xlist[-1])
```

- (a) [-1]
- (b) 4  $\Leftarrow$  **ANSWER**
- (c) [4]
- (d) [2, 3]
- (e) None of the above.

3. [4] What output is produced by the following code?

```
dddd = 4
ylist = ['a', 'bb', 'ccc', dddd, 7, 32]
print(ylist[1 : -1])
```

- (a) ['bb', 'ccc', dddd, 7]
- (b) ['bb', 'ccc', 4, 7]  $\Leftarrow$  **ANSWER**
- (c) ['a', 'bb', 'ccc', 4, 7]
- (d) [32, 7, 4, 'ccc', 'bb']

4. [4] What output is produced by the following code?

```
dddd = 4
ylist = ['a', 'bb', 'ccc', dddd, 7, 32]
print(ylist[ : : -1])
```

- (a) [32, 7, 4, 'ccc', 'bb', 'a']  $\Leftarrow$  **ANSWER**
- (b) [32, 7, dddd, 'ccc', 'bb', 'a']
- (c) ['a', 'bb', 'ccc', 4, 7]
- (d) ['a', 'bb', 'ccc', dddd, 7]

5. [4] What is a general expression for the first element in the sequence `s`?

- (a) `s[-len(s)]` ⇐ **ANSWER**
- (b) `s[-len(s) - 1]`
- (c) `s[1]`
- (d) `s[]`
- (e) None of the above.

6. [4] What output is produced by the following code?

```
xlist = [10, 20]
ylist = xlist
ylist[1] = 44
print(xlist)
```

- (a) `[10, 20]`
- (b) `[44, 20]`
- (c) `[10, 44]` ⇐ **ANSWER**
- (d) `44`
- (e) None of the above.

7. [4] What output is produced by the following code?

```
s = "washington"
s[1] = "W"
print(s)
```

- (a) Washington
- (b) wWshington
- (c) W
- (d) washington ⇐ **Partial Credit**
- (e) This code produces an error. ⇐ **ANSWER**

8. [4] Given that the ASCII value of 'A' is 65, what output is produced by the following code?

```
s = "BCD"
for ch in s:
    print(ord(ch), end=" ")
```

- (a) B C D
- (b) BCD
- (c) 666768 ⇐ **ANSWER**
- (d) 65 66 67
- (e) None of the above.

9. [4] Given that the ASCII value of 'A' is 65, what output is produced by the following code?

```
zlist = [66, 67, 68, 69, 70]
s = ""
for i in range(3):
    s = s + chr(zlist[i])
print(s)
```

- (a) BC
- (b) BCD  $\Leftarrow$  **ANSWER**
- (c) 666768
- (d) ABCDE
- (e) BCDEF
- (f) None of the above.

10. [4] Given that the ASCII value of 'A' is 65, what output is produced by the following code?

```
zlist = [66, 67, 68, 69, 70]
s = ""
for i in zlist:
    s = s + chr(i - 1)
print(s)
```

- (a) BC
- (b) BCD
- (c) 65666768
- (d) ABCDE  $\Leftarrow$  **ANSWER**
- (e) BCDEF
- (f) None of the above.

11. [4] Given that the first printable ASCII character is a space with a value of 32 and the last printable character is tilde with a value of 126, which of the following will show all the printable characters together with the characters' ASCII values?

- (a) 

```
for i in range(32, 126):
    print(i, ':', ord(i), sep="", end=" ")
```
- (b) 

```
for i in range(32, 126):
    print(i, ':', chr(i), sep="", end=" ")
```
- (c) 

```
for i in range(ord( ), ord(~)):
    print(i, ':', chr(i), sep="", end=" ")
```

(d)

```
for i in range(32, 127):
    print(i, ':', chr(i), sep="", end=" ")
```

12. [4] The following code is executed.

```
x = 7
first = 3
offset = 8
y = ((x - first) + offset) % 10 + first
```

What is the resulting value of  $y$ ?

$y =$  5

13. [4] What output is produced by the following code?

```
s = "Lovely Letters"
print(s.count("e"))
```

Output: 3

14. [4] What output is produced by the following code?

```
s = "Lovely Letters"
s.replace("s", "ing.")
print(s)
```

Output: Lovely Letters

15. [4] What output is produced by the following code?

```
s = '110'
count = 0
for b in s[ :: -1]:
    print(int(b) * 2 ** count, end=" ")
    count = count + 1
```

- (a) 1 2 0
- (b) 0 2 4  $\Leftarrow$  **ANSWER**
- (c) 0 1 2
- (d) 2 4 0

For problems 16 and 17, assume the file `foo.txt` contains the following:

---

```
This is  
a test.  
Isn't it? I think so.
```

---

16. [4] What output is produced by the following?

```
file = open("foo.txt", "r")  
file.readline()  
print(file.readline(), end="")
```

Output:  a test.

17. [4] What output is produced by the following?

```
file = open("foo.txt", "r")  
s = file.read()  
print(s[1], end="")
```

Output:  h

18. [4] What output is produced by the following code?

```
x = [['b', 'b', 'b'], ['u', 'y', 'u'], ['y', 'e', 't']]  
def f(z, n):  
    for i in range(len(z)):  
        print(z[i][n], end="")  
    print()
```

`f(x, 1)`

Output:  bye

19. [8] A programmer is asked to write a function called `count_upper()` which takes a single string argument and returns the total number of occurrences of uppercase letters in the string. The programmer attempts to do this by cycling through all the uppercase letters and summing the number of occurrences of each within the given string. Shown below is the programmer's first attempt at writing this function. There is a mistake in each line except the first and third.

```
1 def count_upper(s):
2     upper = ABCDEFGHIJKLMNOPQRSTUVWXYZ
3     sum = 0
4     for ch in range(upper):
5         sum = s.count(ch)
6     return sum
```

In the space provided below, write the correct code for this function (indent as appropriate).

Line 1: `def count_upper(s):`  
Line 2:     upper = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ' # Missing quotes.  
Line 3: `sum = 0`  
Line 4:     for ch in upper: # Do not want range().  
Line 5:         sum = sum + s.count(ch) # Must concatenate with old value.  
Line 6:     return sum # return statement not in loop.

The following demonstrates the behavior of this function after all the code has been corrected.

```
1 >>> count_upper('Hello')
2 1
3 >>> count_upper('HELLO')
4 5
5 >>> count_upper('WoW')
6 2
7 >>> count_upper('WoW!!!')
8 2
```

20. [10] Write a function called `ascii_sum()` which takes a string as its one argument. The function returns an integer that is the sum of the ASCII values for all the characters in the string. For example, if the string 'A B' is passed to this function, the return value is 163 because the ASCII values for A, space, and B are 65, 32, and 66, respectively. (Hint: You will need an accumulator for the sum and a loop to iterate over all the characters of the string.)

**Answer:**

```
1 def ascii_sum(s):
2     sum = 0
3     for ch in s:
4         sum += ord(ch)
5     return sum
```

The following demonstrates correct behavior of this function:

```
1 >>> ord('A')           # First, confirm ASCII value of 'A'.
2 65
3 >>> ascii_sum('A')
4 65
5 >>> ascii_sum('A ')
6 97
7 >>> ascii_sum('A B')
8 163
9 >>> ascii_sum('Hello World!')
10 1085
```

21. [10] Write a function called `splay()` which takes a `list` as its single argument. Every element of this `list` is itself a `list`. `splay()` prints the elements of the inner `lists`, one per line, together with a count of all the elements. The count starts at 1. The output at the bottom of the page demonstrates the correct behavior of this function. (Hint: Think nested `for-loops`.)

**Answer:**

```
1 def splay(xlist):
2     count = 1
3     for outer in xlist:
4         for inner in outer:
5             print(count, inner)
6             count += 1
```

Or, here is an alternate solution that uses explicit indexing:

```
1 def splay(xlist):
2     count = 1
3     for i in range(len(xlist)):
4         for j in range(len(xlist[i])):
5             print(count, xlist[i][j])
6             count = count + 1
```

The following demonstrates correct behavior of this function:

```
1 >>> xlist = [['a', 'b'], [17], [-1.2, 4.5, 10.2]]
2 >>> splay(xlist)
3 1 a
4 2 b
5 3 17
6 4 -1.2
7 5 4.5
8 6 10.2
9 >>> ylist = [[9, 10, ['a', 2], 14], ['hi', 'there', 'fabulous poodles']]
10 >>> splay(ylist)
11 1 9
12 2 10
13 3 ['a', 2]
14 4 14
15 5 hi
16 6 there
17 7 fabulous poodles
```