Sample Midterm

1. true\_count = 0

false\_count = 0

if true\_count < 0: \*This is false

true\_count -= 1

false\_count += 1

else: \*This would run as a result, so “true\_count = -2” and “false\_count = 2”

true\_count -= 2

false\_count += 2

if true\_count > false\_count: \*This is false

true\_count -= 4

false\_count += 4

elif true\_count == false\_count: \*This is also false

true\_count -= 8

false\_count += 8

else: \*This would run as a result, so “true\_count = -18” and “false\_count = 18”

true\_count -= 16

false\_count += 16

print(true\_count, false\_count)

Answer: C

1. doors = ['hello', 'I', 'love', 'you']

remix = doors[-1] + doors[1] + doors[-2] \*This takes the last element, the second element, and the third element

print(remix)

Answer: C

1. polonius = 'This above all'

print(polonius[1:2]) \*The splice of this string would apply to the characters, not the individual words. So this would result in ‘h’ since index 1 points to the second element.

Answer: E

1. import turtle

s = turtle.Screen()

t = turtle.Turtle()

for i in range(4): \*If it weren’t for the extra forward method, this would create a square. The first iteration of the loop would create two sides of a square, but it will create an open polygon.

t.forward(100)

t.right(90)

t.forward(100)

Answer: E

1. day = 'Monday'

mix\_type = [['Every dog'], 'must have', 'his', day]

print(mix\_type[-1]) \*This statement prints the last element in “mix\_type”. This refers to the variable “day” which contains the value ‘Monday’. Therefore, ‘Monday would print.

Answer: E

1. def rithmetic(n1, n2):

for i in range(2):

r = n1%n2 \*Assigns the remainder of division between two integers to r

q = n1//n2 \*Assigns the whole number of division between two integers to r

n1 = r

n2 = q

return (n1,n2)

result = rithmetic(10,3) \*First iteration of the loop returns (1,3) and the second returns (1,0)

print(result)

Answer: C

1. warming = True

highground = True

if or if not warming: \*This is the reason why we have if/else/elif statements. This kind of statement isn’t proper syntax.

print('sun will rise')

if warming:

print("many cat5's")

elif not warming and highground:

print('no flooding')

Answer: A

1. test = 'common'

repeats = 0

for letter in test: \*This will run, but indexing is the only to actually refer to the individual characters of a string. ‘Letter’ simply creates a variable that keeps track of the loop.

if test.count(letter) > 1:

repeats += test.count(letter)

print(repeats)

Answer: B

1. def operate(aSeq, aVal):

if aVal in aSeq: \*This is true

return aVal \*The function ends here because of the return statement

if [aVal] in aSeq:

return [aVal]

if str(aVal) in aSeq:

return str(aVal)

if [str(aVal)] in aSeq:

return [str(aVal)]

zeroes = [0, [0], '0', ['0']]

print(operate(zeroes, 0))

Answer: D

1. def test(seq):

prev = ''

accum = ''

for current\_item in seq: \*The “current\_item iterates through all of the characters of seq

if current\_item in prev:

accum += current\_item

prev += current\_item \*Because of this statement, any character that repeats itself should be added to accum” after its first appearance.

return accum

print(test('hallelujah'))

Answer: D

1. def double\_circle(t, radius):

t.circle(radius)

t.penup()

t.forward(2\*radius)

t.pendown()

t.circle(radius)

t.penup()

t.rt(180)

t.forward(2\*radius)

t.rt(180)

B) def circle\_line(t, radius, multiplier, num\_repeats, separation):

for I in range(num\_repeats):

double\_circle(t, radius)

radius\*=multiplier

t.penup()

t.right(90)

t.forward(separation)

t.left(90)

t.pendown()