One Hundred (or so!) Problems covering CS0 Concepts

Name and User ID:

Choose the best answer (unless otherwise indicated). The language of this quiz is Python 3.x.

These concepts comprise part of the prerequisite knowledge for 100P:

- Literals, expressions, and types
- Variables
- Assignment
- Conditional statements
- Functions and parameters
- Recursion
- Looping

Major Concept: Literals and Variables

Concept: *recognizing literals*

1. The following is a literal: `"Hello World!"`
   (a) True
   (b) False
2. The following is a literal: `123e4`
   (a) True
   (b) False
3. The following is a literal: `true`
   (a) False
   (b) True
4. The following is a literal: `True`
   (a) False
   (b) True
5. Which of the following is a literal?
   (a) `True`
   (b) `my_literal`
   (c) `false`
   (d) `_334`
6. Which of the following is a literal?
   (a) `3.19e4`

(b) `_False`

(c) `return`

(d) `true`

(e) `string`

7. Which of the following is a literal?

   (a) `False`

   (b) `else`

   (c) `_string_`

   (d) `literal`

8. Which of the following is not a literal?

   (a) `7`

   (b) `"Awesome"`

   (c) `my_literal`

   (d) `3.2e1`

9. Which of the following is not a literal?

   (a) `literal`

   (b) `2.79`

   (c) `"Ginormous"`

   (d) `False`

Concept: *lists*

10. In the list below, at what index does *cat* reside?

```
["cat", "dog", "turtle"]
```

   (a) 3

   (b) 0

   (c) 2

   (d) 1

   (e) 4

11. What is the second element of this list: `[4, 1, 0]`?

   (a) 4

   (b) 1

   (c) there is no second element

   (d) 0

12. What is the index of the 4 in this list: `[4, 1, 0]`?

   (a) 3

   (b) 1

   (c) 0

   (d) 2

   (e) 4

13. How many elements are in this list: `[4, 1, 0]`?

    (a) 3
    (b) 1
    (c) 0
    (d) 4

14. What is the resulting of adding these two lists:

```
[4, 2, 0] + [3, 1]
```

    (a) `[4, 2, 0, 3, 1]`
    (b) `[4, 3, 2, 1, 0]`
    (c) an error, the lists are not the same size
    (d) `[0, 1, 2, 3, 4]`

15. What element is at index 6 in the list resulting from this addition:

```
[5, 0, 4] + [2, 3, 8]
```

    (a) 2
    (b) 3
    (c) 4
    (d) 0
    (e) there is no index 6 in the resulting list

16. What is the value of a when this code fragment is executed:

```
b = ["hello", 3, True]
a = b[-3]
```

    (a) `True`
    (b) `"hello"`
    (c) `3`

17. What is the value of a when this code fragment is executed:

```
b = ["hello", 3, True]
a = b[0:1]
```

    (a) an error, since an array index is too large
    (b) `"hello"`
    (c) `["hello", 3, True]`
    (d) `["hello", 3]`
    (e) `["hello"]`

18. What is the value of a when this code fragment is executed:

```
b = ["hello", 3, True]
a = b[1:2]
```

    (a) `[3]`
    (b) `[]`
    (c) `[3, True]`

(d) an error, since index 2 is too large

(e) `3`

19. What is the value of *a* when this code fragment is executed:

```
b = ["hello", 3, True]
a = b[2:]
```

(a) `["hello", 3, True]`

(b) `[]`

(c) `[3, True]`

(d) `[True]`

(e) an error, since index 2 is too large

Concept: *operations*

20. Which is *not* a legal operation in Python?

(a) `12.9 + 3`

(b) `"12" + 3`

(c) `13.3 + 8`

(d) `"12" + "3"`

21. Which is *not* a legal operation in Python?

(a) `9 - 7.8`

(b) `"123" + "456" + "7"`

(c) `1 ** 2`

(d) `"4" - "5"`

22. Which is *not* a legal operation in Python?

(a) `"5" < 3`

(b) `3.4 > 7`

(c) `"7" * 3`

(d) `"123" + "654"`

Concept: *recognizing variables*

23. The following is a variable: `"Hello CS150!"`

(a) True

(b) False

24. The following is a variable: HelloCS150

(a) True

(b) False

25. The following is a variable: 1e4

(a) True

(b) False

26. Which of the following is *not* a legal variable name?

(a) `14_national_championships`
(b) `main`
(c) `sum`
(d) `X_Y`

27. Which of the following is *not* a legal variable name?

(a) `Return`
(b) `i`
(c) `_main`
(d) `def`

28. Which of the following is *not* a legal variable name?

(a) `sumOfAll`
(b) `and`
(c) `_main`
(d) `__X_Y`

29. Which of the following is a legal variable name?

(a) `def`
(b) `if`
(c) `1TwoThree`
(d) `And`

30. Which is *not* a legal variable name?

(a) `1_var`
(b) `_underscore`
(c) `reals`
(d) `c0nif3r`

31. Which is a legal variable name?

(a) `False`
(b) `2.1`
(c) `2`
(d) `_`

Concept: *spotting variables*

32. What variable (if any) is *defined* in the following code:

```
def printHi(): print("Hi There!")
```

(a) no variable is defined
(b) `def`
(c) `printHi`
(d) `print`

33. What variable (if any) is *defined* in the following code:

```
def show(): print("Show me!")
```

(a) no variable is defined
(b) `def`
(c) `show`
(d) `print`

34. What variables are *present* in the following code:

```
def printHi(name): print("Hi There,",name)
```

(a) `def` and `name`
(b) `print`
(c) no variable is defined
(d) `printHi`, `print`, and `name`

35. What variables are *present* in the following code:

```
def show(me): print("Hi There,",me)
```

(a) `show`, `print`, and `me`
(b) no variable is defined
(c) `print`
(d) `def` and `show`

36. How many variables are *defined* in the following code?

```
greeting = "Hi There,"

def printHi(name): print(greeting,name)

printHi("Sally")
```

(a) one
(b) three
(c) none
(d) two

37. How many variables are *defined* in the following code?

```
greeting = "Hi There,"

def show(me,last): print(greeting,name,last)

show("Sally","Poe")
```

(a) two
(b) one
(c) four
(d) none

38. How many variables are *defined* in the following code?

```
def main():
    x = 3
    y = "Python"
    return y+str(x)

main()
```

(a) one

(b) four

(c) three

(d) two

39. What is the value is of x after this code fragment is executed?

```
x = 6
y = x
y = y + 1
```

(a) 6

(b) 5

(c) an error results

(d) 7

40. What is the value is of y after this code fragment is executed?

```
x = 6
y = x
x = x + 1
```

(a) 6

(b) an error results

(c) 7

(d) 5

41. What is the value is returned by the function *foo*?

```
def foo():
   x = 5
   y = x
   x = x + 1
   return y
```

(a) 5

(b) 6

(c) 4

(d) an error results

42. Which of the following expressions doubles the value of the variable x and stores the resulting value back in x?

(a) `x * 2`

(b) `x = x * 2`

(c) `x = x ** 2`

(d) `y = x ** 2`

43. Which of the following expressions squares the value of the variable X and stores the resulting value back in X?

(a) `X = X * 2`

(b) `X = X * (1/.5)`

(c) `X = X ** 2`

(d) `X ** 2`

44. Which of the following expressions does *not* cube the value of the variable X and stores the resulting value back in X?

    (a) `X = X ** 3`

    (b) `X = X * X * X`

    (c) `X = (X ** 2) * X`

    (d) `X ** 3`

45. The following code correctly swaps the values of a and b.

```
a = b
b = a
```

    (a) False

    (b) True

46. The following code correctly swaps the values of a and b.

```
temp = a
a = b
b = temp
```

    (a) True

    (b) False

47. The following code correctly swaps the values of a and b.

```
temp = a
b = a
b = temp
```

    (a) True

    (b) False

48. Which set of expressions does *not* correctly swap the values of a and b? Mark all answers that apply.

    (a) `a = b; b = a`

    (b) `temp = a; b = temp; a = b`

    (c) `temp = b; b = a; a = temp`

    (d) `temp = a; a = b; b = temp`

49. Is this code legal?

```
def printHi():
   print("Hi There!")

printHi = 5

print(printHi)
```

    (a) Yes

    (b) No

50. What is the output of the following code:

```
def printHi():
   print("Hi There!")

printHi = 5

printHi()
```

(a) no output, an error occurs

(b) `Hi There!5`

(c) `Hi There!Hi There!Hi There!Hi There!Hi There!`

(d) `5Hi There!`

51. What is the output of the following code:

```
def show():
   print("Hi There!")

show = "print"

print(show)
```

(a) `<function show @ 192817392>`

(b) no output, an error occurs

(c) `print`

(d) `def show(): show("Hi There!")`

52. What is the output of the following code:

```
print = "hello, world!"

print()
```

(a) `<string "hello, world!">`

(b) no output, an error occurs

(c) `hello, world!`

(d) `print`

Major Concept: Conditionals

Concept: *comparisons*

53. Which of the following is *not* a valid Boolean expression?

(a) `x != y`

(b) `x = y`

(c) `x > y`

(d) `x <= y`

54. Which of the following is a valid Boolean expression?

(a) `x + y`

(b) `x != y`

(c) `x = y`

(d) `x += y`

55. Which operator returns `False` when two things are *not* equal.

    (a) `==`
    (b) `!=`
    (c) `<>`
    (d) `=`
    (e) `^=`

56. Which operator returns `False` when two things are *not* equal.

    (a) `==`
    (b) `^=`
    (c) `!=`
    (d) `<>`

57. Which operator returns `True` when two things are equal?

    (a) `<=>`
    (b) `==`
    (c) `=`
    (d) `!=`

58. What does the `==` operator do?

    (a) creates a new variable
    (b) creates (or updates) a new Boolean variable
    (c) compares two things for equality
    (d) gives a new value (but the variable must already exist)

Concept: *combining Boolean values with* `and`, `or`, and `not`

59. What is the value of *a*?

```
y = 4
x = 2
a = y < x or y % 2 == 0
```

    (a) False
    (b) True

60. What is the value of *a*?

```
y = 3
x = 4
a = y < x or y % 2 == 0
```

    (a) True
    (b) False

61. What is the value of *a*?

```
y = 3
x = 2
a = y < x and not(y % 2 == 0)
```

(a) True

(b) False

62. What is the value of a?

```
y = 1
x = 3
a = y < x or not(y % 2 == 0)
```

(a) False

(b) True

63. What is the value of a?

```
y = 3
x = 2
a = not(y < x or y % 2 == 0)
```

(a) False

(b) True

64. What is the value of a?

```
y = 2
x = 3
a = not(y < x or y % 2 == 0)
```

(a) True

(b) False

Concept: *if statements*

65. Consider this code:

```
if P:
    print('A')
else:
    print('B')
```

Mark the true statements. More than one answer may apply.

(a) It is possible that neither `A` nor `B` will be printed

(b) If P is `False`, `B` must printed

(c) If P is `True`, it is possible that `B` is printed

(d) It is possible that both 'A' and `B` will be printed

66. Consider this code:

```
if P:
    print('A')
if Q:
    print('B')
```

Mark the true statements. More than one answer may apply.

(a) It is possible that neither `A` nor `B` will be printed

(b) If P is `True`, `B` might be printed

(c) If P is `True`, it is possible that `B` is printed

(d) It is possible that both `A` and `B` will be printed

(e) If P is `False`, `B` must printed

67. Consider the following code:

```
if P:
    print('A')
elif Q:
    print('B')
else:
    print('C')
```

Mark the true statement(s). More than one answer may apply.

(a) if Q is true, then `B` will be printed, regardless of the value of P

(b) `C` will be printed when both P and Q are False

(c) One of `B` or `C` will always be printed.

(d) The value of Q has no bearing on whether `A` is printed

(e) `A` and `B` cannot both be printed

68. Regarding the following code, which statement(s) are true? More than one answer may apply.

```
if P:
    print('A')
if Q:
    print('B')
else:
    print('C')
```

(a) `B` will be printed when Q is `True`

(b) The value of P has no bearing on whether `B` or `C` is printed

(c) `C` will be printed when Q is `False`

(d) `A` and `B` cannot both be printed

69. Consider the following code:

```
if P:
    print('A')
elif Q:
    print('B')
else:
    print('C')
```

Mark the true statements. More than one answer may apply.

(a) at least one value is printed

(b) at most two values are printed

(c) at most one value is printed

(d) at most three values are printed

(e) at least two values are printed

(f) at least three value are printed

70. Consider the following code:

```
if P:
    print('A')
if Q:
    print('B')
else:
    print('C')
```

Mark the true statements. More than one answer may apply.

(a) at least one value is printed

(b) at most three values are printed

(c) at least two values are printed

(d) at most one value is printed

(e) at most two values are printed

(f) at least three value are printed

71. Consider the following code:

```
if P:
    print('A')
if Q:
    print('B')
if R:
    print('C')
```

Mark the true statements. More than one answer may apply.

(a) at most one value is printed

(b) at least two values are printed

(c) at most three values are printed

(d) at least three value are printed

(e) at most two values are printed

(f) at least one value is printed

72. What is the possible output of this code fragment?

```
x = eval(input("give me a number: "))
if x % 2 == 0:
    print("even")
print("done")
```

(a) `even`

(b) `done` then `even`

(c) nothing is printed

(d) `even` then `done` or just `done`

73. What is the possible output of this code fragment?

```
x = eval(input("give me a number: "))
if x % 2 == 0:
    print("even")
else
    print("odd")
```

(a) `even`

(b) *even* or *odd*, but not both

(c) `odd`

(d) both *even* and *odd*

(e) nothing will print

74. What is *not* a possible output of this code fragment?

```
x = eval(input("give me a number: "))
if x < 100:
    print("smaller")
if x < 1000:
    print("bigger")
```

(a) `smaller`

(b) all outcomes are possible

(c) nothing is printed

(d) both `smaller` and `bigger` are printed (in that order)

(e) `bigger`

Major Concept: Functions

Concept: *parts of a function*

75. What is the signature of this function?

```
def almostSquare(x,y):
    return (x - 1) * (y + 1)
```

(a) `def almostSquare(x,y):`

(b) `return (x - 1) * (y + 1)`

(c) `def almostSquare():`

(d) `def almostSquare(x,y): return times`

76. What is the signature of this function?

```
def f():
    return 1
```

(a) `def f():`

(b) it has no signature

(c) `def f(1):`

(d) `def f(): return 1`

77. What is the body of this function?

```
def almostSquare(x,y):
    return (x - 1) * (y + 1)
```

(a) `return times`

(b) `return (x - 1) * (y + 1)`

(c) `def almostSquare(x,y):`

(d) `def almostSquare():`

78. What is the body of this function?

```
def f(x):
    return g(x * 2)
```

(a) `return g`

(b) `return g(x * 2)`

(c) `def f(g(x * 2))`

(d) `return function call`

Concept: *recognizing function definitions*

79. This is a function definition:

```
square(x)
```

(a) False

(b) True

80. This is a function definition:

```
def square(x):
    return x * x
```

(a) False

(b) True

81. What is this?

```
def almostSquare(x):
    return (x + 1) * (x - 1)
```

(a) a function call

(b) a function definition

(c) a complete program

(d) a loop

Concept: *recognizing function calls*

82. This is a function call:

```
square(x)
```

(a) False

(b) True

83. This is a function call:

```
def square(x):
    return x * x
```

(a) False

(b) True

84. What is this?

```
almostSquare(a)
```

(a) a loop
(b) a function call
(c) a complete program
(d) a function definition

Concept: *recognizing complete programs*

85. What is this?

```
def almostSquare(x):
    return (x + 1) * (x - 1)

print(almostSquare(5))
```

(a) a complete program
(b) a function call
(c) a loop
(d) a function definition

86. Is this a complete program?

```
def main():
    x = 5
    print("x is",x)
```

(a) no, *main* is called, but not defined
(b) no, *main* is defined, but not called
(c) yes, the program prints correctly

87. Is this a complete program?

```
def main():
    x = 5
    print("x is",x)

main()
```

(a) yes, the program prints correctly
(b) no, *main* is defined, but not called
(c) no, *main* is called, but not defined

88. Is this a complete program?

```
def main():
    x = 5
    print("x is",x)

    main()
```

(a) no, *main* is called, but not defined

(b) yes, the program prints correctly

(c) no, *main* is defined, but not called

(d) no, *main* is called, but only from within *main*

Concept: *recognizing errors in function definitions*

89. Is this function definition correct?

```
def raise(x,y)
    return x ** y
```

(a) no, a colon character is missing

(b) yes, it is correct as written

(c) no, the return statement should not be indented

(d) no, there should only be one formal parameter

90. Is this function definition correct?

```
def raise(x,y):
return x ** y
```

(a) no, the return statement should be indented

(b) yes, it is correct as written

(c) no, there should only be one formal parameter

(d) no, there should be two colon characters

91. Is this function definition correct?

```
def raise(x,y):
    return x ** y
```

(a) No

(b) Yes

92. Is this function definition correct?

```
define raise(x,y):
    return x ** y
```

(a) no, there should only be one formal parameter

(b) yes, it is correct as written

(c) no, the keyword is `def`, not `define`

(d) no, there should be two colon characters

(e) no, the return statement should not be indented

93. Is this function definition correct?

```
def raise(x,y):
    x ** y
```

(a) yes, it is correct as written

(b) no, there should only be one formal parameter

(c) no, there should be two colon characters

(d) no, there should be a `return`

94. Is this function definition correct?

```
def raise(x y):
    return x ** y
```

(a) no, there should be a comma between formal parameters
(b) yes, it is correct as written
(c) no, there should only be one formal parameter
(d) no, there should be two colon characters

Concept: *matching calls and definitions*

95. Does the function call match the function definition?

```
def square(x):
    return x * x

result = square()
```

(a) no, there are too few formal parameters
(b) no, there are too few arguments
(c) no, there are too many arguments
(d) yes

96. This function call matches the function definition:

```
def square(x):
    return x * x

result = square()
```

(a) False
(b) True

97. Does the function call match the function definition?

```
def square(x):
    return x * x

result = square(3)
```

(a) no, there are too many arguments
(b) yes
(c) no, there are too few formal parameters
(d) no, the function call should pass the variable x

98. This function call matches the function definition:

```
def square(x):
    return x * x

result = square(3)
```

(a) False
(b) True

99. Does the function call match the function definition?

```
def square(x):
    return x * x

a = 3
result = square(a)
```

(a) yes

(b) no, the function call should pass the variable x, not a

(c) no, the function call should pass a value, not a variable

(d) no, there are too few formal parameters in the definition

(e) no, there are too many arguments in the call

100. This function call matches the function definition:

```
def square(x):
    return x * x

a = 3
result = square(a)
```

(a) True

(b) False

101. Does the function call match the function definition?

```
def square(x):
    return x * x

result = square(3,7)
```

(a) no, there are too many arguments in the call

(b) no, there are too few formal parameters in the definition

(c) yes

(d) no, the function call should pass the variable x twice

102. This function call matches the function definition:

```
def square(x):
    return x * x

result = square(3,7)
```

(a) False

(b) True

103. Does the function call match the function definition?

```
def almostProduct(a,b):
    return a * b

x = 3
y = 7
result = AlmostProduct(x,y)
```

(a) no, the function call should pass the variables a and b

(b) yes

(c) no, the function names do not match

(d) no, the function call should pass two values

104. This function call matches the function definition:

```
def almostProduct(a,b):
    return a * b

x = 3
y = 7
result = AlmostProduct(x,y)
```

(a) True

(b) False

105. Does the function call match the function definition?

```
def almostProduct(a,b):
    return x * x

x = 3
y = 7
result = almostProduct(x,y,x * x)
```

(a) no, there are too many formal parameters

(b) no, the function call should pass values

(c) yes

(d) no, there are too many arguments

(e) no, the function names do not match

106. This function call matches the function definition:

```
def almostProduct(a,b):
    return x * x

x = 3
y = 7
result = almostProduct(x,y,x * x)
```

(a) False

(b) True

Concept: *identifying arguments*

107. Identify the function call arguments in the code below:

```
def compute(x,y):
    a = x + 1
    b = y - 1
    return a * b

j = 3
k = 7
result = compute(j + 1,k - 1)
```

(a) the variables j and k

(b) the variables x and y

(c) the expressions involving variables j and k

(d) the values *3* and *7*

108. What are the variables j and k in the code below:

```
def compute(x,y):
    a = x + 1
    b = y - 1
    return a * b

j = 3
k = 7
result = compute(j,k)
```

(a) local variables defined in the body of the it compute function

(b) the arguments given in a function call

(c) the formal parameters of a function

109. What are the formal parameters of the *compute* function?

```
def compute(x,y):
    a = x + 1
    b = y - 1
    return a * b

j = 3
k = 7
result = compute(j,k)
```

(a) a, b

(b) j, k

(c) the compute function has no formal parameters

(d) x, y

110. What is being bound to the formal parameters? Choose the most precise answer.

```
def compute(x,y):
    a = x + 1
    b = y - 1
    return a * b

j = 3
k = 7
result = compute(j,k)
```

(a) the variables x and y

(b) the values of variables j and k

(c) the variables j and k

(d) the values of variables x and y

Concept: *analyzing code*

111. What is the value of x while the function body is being evaluated?

```
def compute(x,y):
    a = x + 1
    b = y - 1
    return a * b

j = 3
k = 7
result = compute(j,k)
```

(a) `k`

(b) `j`

(c) 3

(d) 7

112. What is the value of y while the function body is being evaluated?

```
def compute(x,y):
    a = x + 1
    b = y - 1
    return a * b

j = 3
k = 7
result = compute(j,k)
```

(a) 7

(b) 3

(c) `k`

(d) `y`

113. What are the arguments, formal parameters, and local variables of the *compute* function?

```
def compute(x,y):
    a = x + 1
    b = y - 1
    return a * b

result = compute(3,7)
```

(a) x and y, x and y, a and b, respectively

(b) 3 and 7, a and b, x and y, respectively

(c) the compute function has no local variables

(d) 3 and 7, x and y, a and b, respectively

114. What is printed by this program:

```
def almostSquare(x,y):
    (x - 1) * (y + 1)

print(almostSquare(5,5))
```

(a) `None`

(b) nothing is printed because of an error

(c) 25

(d) 24

115. What is printed by this program:

```
def almostSquare(x,y):
    (x - 1) * (y + 1)

x = almostSquare(5,5)
print(x)
```

(a) `None`

(b) nothing is printed because of an error

(c) 24

(d) 25

Concept: *comments*

116. The code below produces what output?

```
#print("Hello, World!",sep="-")
```

(a) (no output, it's a comment)

(b) `Hello- World!`

(c) `Hello-World!`

(d) `Hello, World!`

117. The code below produces what output?

```
print("Hello, World!",sep="-")
```

(a) `Hello, World!-`

(b) `Hello, World!`

(c) (no output, it's a comment)

(d) `Hello,-World!`

118. Which character starts a comment in a Python program?

(a) `%`

(b) `&`

(c) `$`

(d) #

119. The # character means that:

(a) Python should escape the following special character in a string

(b) Python should treat the remainder of the line as a string

(c) Python should ignore it and any remaining characters on the line

(d) Python should hash the following string into a number

Major Concept: Scope

Concept: *vocabulary*

120. With respect to scope, another name for 'visible' is:

(a) in scope
(b) out of scope

121. With respect to scope, another name for 'inaccessible' is:

(a) out of scope
(b) in scope

122. With respect to scope, another name for 'accessible' is:

(a) in scope
(b) out of scope

Concept: *enclosing scopes*

123. Variables in an *enclosing* scope are:

(a) in scope
(b) out of scope

124. Variables in an *enclosed* scope are:

(a) out of scope
(b) in scope

125. The global scope *encloses* all other scopes.

(a) False
(b) True

126. Every other scope *encloses* the global scope.

(a) False
(b) True

127. Suppose in the body of function f, function g is defined. The scope of g:

(a) encloses the scope of f
(b) is enclosed by the scope of f

Concept: *scopes and definitions*

128. The set of variables *defined* in the scope of a function body includes the local variables.

(a) True
(b) False

129. The set of variables *defined* in the scope of a function body includes the non-local variables.

(a) True
(b) False

130. With respect to a function body, the function name belongs to the:

(a) set of variables that are out of scope
(b) set of local variables
(c) set of non-local variables

131. With respect to a function body, the function name is defined in:

(a) the scope that holds the function definition

(b) the scope enclosing the scope that holds the function definition

(c) the scope of the function body

132. The set of variables *defined* in the scope of a function body includes the name of the function.

(a) True

(b) False

133. With respect to a function body, the formal parameters of the function belong to the:

(a) set of non-local variables

(b) set of local variables

(c) set of variables that are out of scope

134. With respect to a function body, the formal parameters of the function are defined in the:

(a) the scope of the function body

(b) the scope enclosing the scope that holds the function definition

(c) the scope that holds the function definition

135. The set of variables *defined* in the scope of a function body includes the formal parameters.

(a) False

(b) True

Concept: modules

136. Suppose module *a.py* has a function named *f* that takes one argument. How would I call this function if *a.py* is imported this way:

```
import a
```

(a) `a.f(x)`

(b) `f(x)`

(c) `a(f,x)`

(d) `a:f(x)`

137. Suppose module *a.py* has a function named *f* that takes one argument. How would I call this function if *a.py* is imported this way:

```
from a import *
```

(a) `a:f(x)`

(b) `a.f(x)`

(c) `a(f,x)`

(d) `f(x)`

138. Suppose module *a.py* has a function *def f(x):*. Suppose the program that imports *a.py* also has an *def f(x):*. How would I call *a.py*'s version of *f* if *a.py* is imported with this statement at the top of the file:

```
from a import *
```

(a) `a:f(z)`

(b) function *f* in *a.py* cannot be called by any of these methods

(c) `a.f(z)`

(d) `f(z)`

139. Suppose module *a.py* has a function *def f(x):*. Suppose the program that imports *a.py* also has an *def f(x):*. How would I call *a.py*'s version of *f* if *a.py* is imported with this statement at the top of the file:

```
import a
```

(a) function *f* in *a.py* cannot be called by any of these methods
(b) `f(a,z)`
(c) `f(z)`
(d) `a.f(z)`

Concept: *undefined variables*

140. Is the following code legal?

```
def main(): y = x

main()
```

(a) yes, this is legal
(b) no, *y* cannot be a variable
(c) no, *x* does not exist
(d) no, the definition of *main* precedes the call to *main*

141. What is the output when *main* is called?

```
def main(): print("x is",x)
```

(a) `x is 0`
(b) `None`
(c) an error message
(d) `x is x`

142. What is the output when *main* is called?

```
x = 3

def main(): print("x is",x)
```

(a) `x is x`
(b) an error message
(c) `x is 0`
(d) `x is 3`

143. What is the output when *main* is called?

```
def main(): print("x is",x)

x = 3
```

(a) an error message, because *x* is undefined in *main*
(b) `x is 3`
(c) an error message, because *main* is defined before *x*

(d) `x is 0`

144. What is the output when *main* is called?

```
def main():
    z = squarePlus(4)
    print("y is",y)

def squarePlus(x):
    y = x + 1
    return y * y
```

(a) `y is 4`
(b) an error message
(c) `y is 0`
(d) `y is 5`

145. What is the output when *main* is called?

```
def main():
    z = squarePlus(4)
    print("z is",z)

def squarePlus(x):
    y = x + 1
    y * y
```

(a) an error message
(b) `y is 5`
(c) `None`
(d) `y is 0`
(e) `y is 4`

146. What is the output when *main* is called?

```
def squarePlus(x):
    y = x + 1
    return y * y

def main():
    z = squarePlus(4)
    print("y is",y)
```

(a) `y is 5`
(b) `y is 4`
(c) an error message, because *squarePlus* is defined before *main*
(d) an error message, because y is not in the scope of *main*
(e) `y is 0`

147. What is the output when *main* is called?

```
def squarePlus(x):
    y = x + 1

def main():
    z = squarePlus(4)
    print("y is",y)
```

(a) `y is 0`

(b) `y is 4`

(c) an error message, because *squarePlus* does not return a value

(d) an error message, because y is not in the scope of *main*

(e) `y is 5`

148. What is the output when *main* is called?

```
def main():
    z = 1831
    print(z)
z = 2031
```

(a) `3862`

(b) `2031`

(c) an error message

(d) `1831`

149. What is the output when *main* is called?

```
z = 2031
def main():
    z = 1831
    print(z)
```

(a) `2031`

(b) an error message

(c) `1831`

(d) 3862

150. What is the output when *main* is called?

```
def b(): value = 100

def a(): value = 12

def main():
    a()
    b()
    print(value)
```

(a) `100`

(b) `None`

(c) an error message

(d) `12`

Concept: *global variables*

151. What is the output when *main* is called?

```
value = 333

def b(): value = 100

def a(): value = 12

def main():
    a()
    b()
    print(value)
```

(a) 333

(b) an error message

(c) 12

(d) 100

152. What is the output when *main* is called?

```
value = 333

def b():
    global value
    value = 100

def a(): value = 12

def main():
    a()
    b()
    print(value)
```

(a) 100

(b) None

(c) 12

(d) 333

(e) an error message

153. What is the output when *main* is called?

```
value = 333

def b(): value = 100

def a():
    global value
    value = 12

def main():
    a()
    b()
    print(value)
```

(a) 333

(b) an error message

(c) 12

(d) 345

(e) `100`

154. What is the output when *main* is called?

```
value = 444

def b():
    global value
    value = 111

def a():
    global value
    value = 66

def main():
    a()
    b()
    print(value)
```

(a) `66`

(b) `444`

(c) `111`

(d) an error message

155. Which function call (if any) causes an error?

```
def apples(count):
    print("you have",count,"apples")

def oranges(amount):
    print("you have",amount,"oranges")

count = 10
apples(count)
oranges(count)
```

(a) neither the call to *apples* nor the call to *oranges*

(b) both the call to *apples* and the call to *oranges*

(c) the call to *apples*

(d) the call to *oranges*

156. Which function call (if any) causes an error?

```
def apples(count,amount):
    print("you have",count + amount,"apples")

def oranges(amount,count):
    print("you have",amount + count,"oranges")

count = 10
amount = 5
apples(count,amount)
oranges(count,amount)
```

(a) neither the call to *apples* nor the call to *oranges*

(b) the call to *apples*

(c) both the call to *apples* and the call to *oranges*

(d) the call to *oranges*

Concept: *two scope levels, inner scope*

157. The variables *visible* in scope 2 are:

```
z = 0           # scope 1
def f(x):
    a = x - 1       # scope 2
    b = x + 1
    return b * b - a * a

z = f(4)
print(z)
```

(a) *a*, *b*

(b) *a*, *b*, *f*, *x*

(c) *a*, *b*, *x*

(d) *a*, *b*, *f*, *x*, *z*

158. The variables *defined* in scope 2 are:

```
z = 0           # scope 1
def f(x):
    a = x - 1       # scope 2
    b = x + 1
    return b * b - a * a

z = f(4)
print(z)
```

(a) *a*, *b*

(b) *a*, *b*, *x*

(c) *a*, *b*, *f*, *x*

(d) *a*, *b*, *f*, *x*, *z*

Concept: *two scope levels, outer scope*

159. The variables *defined* in scope 1 are:

```
z = 0           # scope 1
def f(x):
    a = x - 1       # scope 2
    b = x + 1
    return b * b - a * a

z = f(4)
print(z)
```

(a) *z*

(b) *z*, *f*

(c) *z*, *f*, *x*

(d) *z*, *f*, *x*, *a*, *b*

160. The variables *visible* in scope 1 are:

```
z = 0           # scope 1
def f(x):
    a = x - 1       # scope 2
    b = x + 1
    return b * b - a * a

z = f(4)
print(z)
```

(a) z, f

(b) z, f, x

(c) z

(d) z, f, x, a, b

161. Variable x is *visible* in which scopes?

```
z = 0           # scope 1
def f(x):
    a = x - 1       # scope 2
    b = x + 1
    return b * b - a * a

z = f(4)
print(z)
```

(a) scope 1

(b) scopes 1 and 2

(c) scope 2

162. Variable f is *visible* in which scopes?

```
z = 0           # scope 1
def f(x):
    a = x - 1       # scope 2
    b = x + 1
    return b * b - a * a

z = f(4)
print(z)
```

(a) scope 1

(b) scope 2

(c) scopes 1 and 2

Concept: *three scope levels, inner scope*

163. The variables *defined* in scope 3 are:

```
z = 0           # scope 1
def f(x):
    a = x - 1       # scope 2
    b = x + 1
    def g(r):
        m = r * x       # scope 3
```

```
            return m * m
        return b * b - a * a + g(a + b)

    z = f(4)
    print(z)
```

(a) m, j, g

(b) x, g

(c) m, g

(d) m, r

164. The variables *visible* in scope 3 are:

```
z = 0           # scope 1
def f(x):
    a = x - 1       # scope 2
    b = x + 1
    def g(r):
        m = r * x       # scope 3
        return m * m
    return b * b - a * a + g(a + b)

z = f(4)
print(z)
```

(a) m, x

(b) m, x, g

(c) m, x, g, a, b

(d) all of them

165. Variable f is *visible* in which scopes?

```
z = 0           #scope 1
def f(x):
    a = x - 1       # scope 2
    b = x + 1
    def g(r):
        m = r * x       #spot 3
        return m * m
    return b * b - a * a + g(a + b)

z = f(4)
print(z)
```

(a) scope 1

(b) scopes 1 and 2

(c) scopes 1, 2, and 3

(d) scopes 2 and 3

166. Variable g is *visible* in which scopes?

```
z = 0           #scope 1
def f(x):
    a = x - 1       # scope 2
    b = x + 1
    def g(r):
```

```
            m = r * x       # scope 3
            return m * m
        return b * b - a * a + g(a + b)

    z = f(4)
    print(z)
```

(a) scopes 1 and 2

(b) scopes 1, 2, and 3

(c) scopes 2 and 3

(d) scope 1

Concept: *three scope levels, middle scope*

167. The variables *visible* in scope 2 are:

```
z = 0           #scope 1
def f(x):
    a = x - 1       # scope 2
    b = x + 1
    def g(r):
        m = r * x       # scope 3
        return m * m
    return b * b - a * a + g(a + b)

z = f(4)
print(z)
```

(a) r, m

(b) all of them, except r, m, and g

(c) all of them, except r and m

(d) all of them

168. The variables *visible* in scope 2 are:

```
z = 0           #scope 1
def f(x):
    a = x - 1       # scope 2
    b = x + 1
    def g(r):
        m = r * x       # scope 3
        return m * m
    return b * b - a * a + g(a + b)

z = f(4)
print(z)
```

(a) x, a, b, g

(b) x, a, b, g, r, m

(c) all of them, except r and m

(d) all of them

Concept: *three scope levels, outer scope*

169. The variables *defined* in scope 1 are:

```
z = 0           #scope 1
def f(x):
    a = x - 1       # scope 2
    b = x + 1
    def g(r):
        m = r * x       # scope 3
        return m * m
    return b * b - a * a + g(a + b)

z = f(4)
print(z)
```

(a) z

(b) z, f

(c) z, f, x, a, b

(d) z, f, x

170. The variables *visible* in scope 1 are:

```
z = 0           #scope 1
def f(x):
    a = x - 1       # scope 2
    b = x + 1
    def g(r):
        m = r * x       # scope 3
        return m * m
    return b * b - a * a + g(a + b)

z = f(4)
print(z)
```

(a) z, f, x

(b) all of them

(c) z

(d) z, f

Concept: *parallel scope levels, inner scopes*

171. The variables *defined* in scope 2a are:

```
z = 0           #scope 1
def f(x):
    a = x - 1       # scope 2a
    return a * a
def g(y):
    b = y + 1       # scope 2b
    return b * b

z = f(4) + g(5)
print(z)
```

(a) a

(b) x, a, y, b

(c) a, b

(d) x, a

172. The variables *defined* in scope $2b$ are:

```
z = 0           #scope 1
def f(x):
    a = x - 1       # scope 2a
    return a * a
def g(y):
    b = y + 1       # scope 2b
    return b * b

z = f(4) + g(5)
print(z)
```

(a) x, a, y, b

(b) a, b

(c) g, y, b

(d) y, b

(e) b

173. The variables *visible* in scope $2a$ are:

```
z = 0           #scope 1
def f(x):
    a = x - 1       # scope 2a
    return a * a
def g(y):
    b = y + 1       # scope 2b
    return b * b

z = f(4) + g(5)
print(z)
```

(a) z, f, x, a

(b) a

(c) z, f, g, x, a

(d) all of them

(e) x, a

174. The variables *visible* in scope $2b$ are:

```
z = 0           #scope 1
def f(x):
    a = x - 1       # scope 2a
    return a * a
def g(y):
    b = y + 1       # scope 2b
    return b * b

z = f(4) + g(5)
print(z)
```

(a) z, g, y, b, f

(b) all of them

(c) *b*

(d) *y*, *b*

Concept: *parallel scope levels, outer scopes*

175. The variables *defined* in scope 1 are:

```
z = 0           #scope 1
def f(x):
    a = x - 1       # scope 2a
    return a * a
def g(z):
    b = x + 1       # scope 2b
    return b * b

z = f(4) + g(5)
print(z)
```

(a) *z*, *f*, *g*

(b) *z*

(c) *z*, *f*, *x*, *g*, *z*

(d) *z*, *f*, *x*, *a*, *g*, *z*, *b*

176. The variables *visible* in scope 1 are:

```
z = 0           #scope 1
def f(x):
    a = x - 1       # scope 2a
    return a * a
def g(z):
    b = x + 1       # scope 2b
    return b * b

z = f(4) + g(5)
print(z)
```

(a) *z*, *f*, *g*

(b) all of them

(c) *z*, *f*, *x*, *g*, *z*

(d) *z*

Major Concept: Recursion

Concept: recognizing *counts* and *accumulations*

177. Which pattern does the following function implement?

```
def f(s):
    if (s == []):
        return 1
    elif isPrime(head(s)):
        return head(s) * f(tail(s))
    else:
        return f(tail(s))
```

(a) the *filtered-counting* pattern

(b) the *accumulate* pattern

(c) the *filtered-accumulate* pattern

(d) the *counting* pattern

178. Which pattern does the following function implement?

```
def f(s):
    if (s == []):
        return 0
    else:
        return 1 + f(tail(s))
```

(a) the *filtered-counting* pattern

(b) the *filtered-accumulate* pattern

(c) the *counting* pattern

(d) the *accumulate* pattern

179. Which pattern does the following function implement?

```
def f(s):
    if (s == []):
        return 0
    elif isPrime(head(s)):
        return 1 + f(tail(s))
    else:
        return f(tail(s))
```

(a) the *counting* pattern

(b) the *filtered-counting* pattern

(c) the *filtered-accumulate* pattern

(d) the *accumulate* pattern

180. Which pattern does the following recurrence implement?

```
g(t) is 0 if t is []
g(t) is 1 + g(tail(t))
    otherwise
```

(a) the *filtered-counting* pattern

(b) the *filtered-accumulate* pattern

(c) the *accumulate* pattern

(d) the *counting* pattern

181. Which pattern does the following recurrence implement?

```
g(t) is 1 if t is the empty list
g(t) is head(t) * g(tail(t))
    if head(t) has property Y
g(t) is g(tail(t)) otherwise
```

(a) the *filtered-counting* pattern

(b) the *filtered-accumulate* pattern

(c) the *counting* pattern

(d) the *accumulate* pattern

182. Which pattern does the following recurrence implement?

```
h(r) is 0 if r is []
h(r) is 1 + h(tail(r))
    if (head(r)) has property Z
h(r) is h(tail(r)) otherwise
```

(a) the *filtered-counting* pattern

(b) the *accumulate* pattern

(c) the *counting* pattern

(d) the *filtered-accumulate* pattern

183. Which pattern does the following recurrence implement?

```
h(r) is 0 if r is []
h(r) is head(r) + h(tail(r))
    otherwise
```

(a) the *accumulate* pattern

(b) the *filtered-counting* pattern

(c) the *counting* pattern

(d) the *filtered-accumulate* pattern

184. Which pattern does the following function implement?

```
def f(s):
    if (s == []):
        return 1
    else:
        return head(s) * f(tail(s))
```

(a) the *filtered-counting* pattern

(b) the *filtered-accumulate* pattern

(c) the *accumulate* pattern

(d) the *counting* pattern

Concept: picking patterns, *counts* and *accumulate*

185. If I wish to solve this problem: *determine the product of the numbers from* a *to* b, I should implement the:

(a) the *counting* pattern

(b) the *filtered-counting* pattern

(c) the *accumulate* pattern

(d) the *filtered-accumulate* pattern

186. If I wish to solve this problem: *count the number of letters in a string* and the string contains letters and digits, I should implement the:

(a) the *filtered-counting* pattern

(b) the *accumulate* pattern

(c) the *filtered-accumulate* pattern

(d) the *counting* pattern

187. If I wish to solve this problem: *count the number of Charmichael numbers in the range* a *to* b, I should implement the:

(a) the *counting* pattern

(b) the *filtered-counting* pattern

(c) the *filtered-accumulate* pattern

(d) the *accumulate* pattern

188. If I wish to solve this problem: *determine the sum of the Charmichael numbers in the range* a *to* b, I should implement the:

(a) the *filtered-counting* pattern

(b) the *filtered-accumulate* pattern

(c) the *counting* pattern

(d) the *accumulate* pattern

189. If I wish to solve this problem: *determine the product of the numbers in a list* and the list contains only numbers, I should implement the:

(a) the *counting* pattern

(b) the *filtered-accumulate* pattern

(c) the *accumulate* pattern

(d) the *filtered-counting* pattern

190. If I wish to solve this problem: *determine the product of the odd numbers in a list*, I should implement the:

(a) the *counting* pattern

(b) the *accumulate* pattern

(c) the *filtered-accumulate* pattern

(d) the *filtered-counting* pattern

191. If I wish to solve this problem: *shuffle two lists, creating a third list*, I should implement the:

(a) the *shuffle* pattern

(b) the *filtered-accumulate* pattern

(c) the *filtered-shuffle* pattern

(d) the *shuffle-counting* pattern

Concept: *recognizing map, filter, and search*

192. The *filter* pattern is a special case of a *filtered-accumulation*.

(a) False

(b) True

193. Which pattern does the following recurrence implement?

```
h(g,t) is [] if t is []
h(g,t) is [g(head(t))] + h(g,tail(t)) otherwise
```

(a) the *search* pattern

(b) the *filter* pattern

(c) the *map* pattern

194. Which pattern does the following function implement?

```
def f(h,r):
    if (r == []):
        return []
    else:
        return [h(head(r))] + f(h,tail(r))
```

(a) the *map* pattern

(b) the *filter* pattern

(c) the *search* pattern

195. Which pattern does the following recurrence implement?

```
g(t) is [] if t is []
g(t) is [head(t)] + g(tail(t))
      if isX(head(t)) is true
g(t) is g(tail(t)) otherwise
```

(a) the *filter* pattern

(b) the *search* pattern

(c) the *map* pattern

196. Which pattern does the following function implement?

```
def g(t):
    if (t == []):
        return []
    elif (isY(head(t))):
        return [head(t)] + g(tail(t))
    else:
        return g(tail(t))
```

(a) the *filter* pattern

(b) the *map* pattern

(c) the *search* pattern

Concept: picking patterns, *map*, *filter*, and *search*

197. If I wish to solve this problem: *square every number in a list*, I should implement the:

(a) the *filter* pattern

(b) the *accumulate* pattern

(c) the *map* pattern

(d) the *search* pattern

198. If I wish to solve this problem: *find if there is an even number in a list*, I should implement the:

(a) the *accumulate* pattern

(b) the *search* pattern

(c) the *map* pattern

(d) the *filter* pattern

199. If I wish to solve this problem: *extract the Charmichael numbers from a list*, I should implement the:

(a) the *search* pattern

(b) the *filter* pattern

(c) the *map* pattern

200. If I wish to solve this problem: *sum the squares of every number in a list*, I should implement the:

(a) the *filter* pattern followed by the *accumulate* pattern

(b) the *map* pattern followed by the *accumulate* pattern

(c) the *count* pattern followed by the *accumulate* pattern

(d) the *filter* pattern followed by the *search* pattern

(e) the *map* pattern followed by the *search* pattern

Concept: *verifying code*

201. Consider the problem statement: *sum the numbers from* a *to* b (*inclusive*). Assuming a is less than or equal to b, does this recursive function compute the correct result?

```
def sum(a,b):
    if (a == b):
        return 0
    else:
        return a + sum(a + 1,b)
```

(a) No

(b) Yes

202. Consider the problem statement: *sum the numbers from* a *to* b (*inclusive*). Assuming a is less than or equal to b, does this recursive function compute the correct result?

```
def sum(a,b):
    if (a == b):
        return a
    else:
        return a + sum(a + 1,b)
```

(a) Yes

(b) No

203. Consider the problem statement: *sum the numbers from* a *to* b (*inclusive*). Assuming a is less than or equal to b, does this recursive function compute the correct result?

```
def sum(a,b):
    if (a == b):
        return b
    else:
        return b + sum(a + 1,b)
```

(a) Yes

(b) No

204. Consider the problem statement: *sum the numbers from* a *to* b (*inclusive*). Assuming a is less than or equal to b, does this recursive function compute the correct result?

```
def sum(a,b):
    if (a == b):
        return b
    else:
        return a + sum(a + 1,b)
```

(a) Yes

(b) No

205. Consider the problem statement: *sum the numbers from* a *to* b (*inclusive*). Assuming a is less than or equal to b, does this recursive function compute the correct result?

```
def sum(a,b):
    if (a == b):
        return 1
    else:
        return a + sum(a + 1,b)
```

(a) No

(b) Yes

206. Consider the problem statement: *sum the numbers from* a *to* b (*inclusive*). Assuming a is less than or equal to b, does this recursive function compute the correct result?

```
def sum(a,b):
    if (a > b):
        return 0
    else:
        return a + sum(a + 1,b)
```

(a) Yes

(b) No

207. Consider the problem statement: *sum the numbers from* a *to* b (*inclusive*). Assuming a is less than or equal to b, does this recursive function compute the correct result?

```
def sum(a,b):
    if (a == b):
        return a
    else:
        return b + sum(a,b-1)
```

(a) Yes

(b) No

208. Consider the problem statement: *sum the numbers from* a *to* b (*inclusive*). Assuming a is less than or equal to b, does this recursive function compute the correct result?

```
def sum(a,b):
    if (a == b):
        return a
    else:
        return a + sum(a,b-1)
```

(a) Yes

(b) No

209. Consider the problem statement: *sum the numbers from* a *to* b (*inclusive*). Assuming a is less than or equal to b, does this recursive function compute the correct result?

```
def sum(a,b):
    if (a > b):
        return 0
    else:
        return b + sum(a,b-1)
```

(a) Yes

(b) No

210. Consider the problem statement: *sum the numbers from* a *to* b (*inclusive*). Assuming a is less than or equal to b, does this recursive function compute the correct result?

```
def sum(a,b):
    if (a > b):
        return b
    else:
        return b + sum(a,b-1)
```

(a) Yes

(b) No

211. Consider the problem statement: *shuffle two lists, creating a third list.* Does this recursive function compute the correct result?

```
def shuffle(list1,list2):
    if (list1 == []):
        return list2
    else:
        return [head(list1)] + \
            shuffle(list2,tail(list1))
```

(a) No, the function can fail with an error

(b) No, the function runs without error, but does not shuffle

(c) Yes

212. Consider the problem statement: *shuffle two lists, creating a third list.* Does this recursive function compute the correct result?

```
def shuffle(list1,list2):
    if (list1 == []):
        return list2
    else:
        return head(list1) + \
            shuffle(list2,tail(list1))
```

(a) No, the function can fail with an error

(b) Yes

(c) No, the function runs without error, but does not shuffle

213. Consider the problem statement: *shuffle two lists, creating a third list.* Does this recursive function compute the correct result?

```
def shuffle(list1,list2):
    if (list1 == []):
        return list2
    else:
        return \
            [head(list1),head(list2)] + \
            shuffle(tail(list1),tail(list2))
```

(a) No, the function can fail with an error

(b) Yes

(c) No, the function runs without error, but does not shuffle

214. Consider the problem statement: *shuffle two lists, creating a third list.* Does this recursive function compute the correct result?

```
def shuffle(list1,list2):
    if (list1 == []):
        return list2
    elif (list2 == []):
        return list1
    else:
        return \
            [head(list1),head(list2)] + \
            shuffle(tail(list1),tail(list2))
```

(a) Yes

(b) No, the function runs without error, but does not shuffle

(c) No, the function can fail with an error

Concept: implementing recurrences

215. Implement this recurrence:

```
f(a,b) is 1 if b is zero
f(a,b) is a * f(a,b - 1) otherwise
```

216. Implement this recurrence:

```
f(t,a,b) is t if b is zero
f(t,a,b) is f(a * t,a,b - 1) otherwise
```

217. Implement this recurrence:

```
f(n) is 0 if n is 0
f(n) is 1 if n is 1
f(n) is f(n - 1) + f(n - 2) otherwise
```

218. Implement this recurrence:

```
g(a,b,n) is a if n is 0
g(a,b,n) is g(b,a + b,n - 1) otherwise
```

219. Implement this recurrence:

```
f(t,a,b) is t if b is zero
f(t,a,b) is f(t,a * a,b / 2) if b is even
f(t,a,b) is f(t * a,a,b - 1) otherwise
```

220. Implement this recurrence:

```
f(a,b) is 1 if b is zero
f(a,b) is f(a * a,b / 2) if b is even
f(a,b) is a * f(a,b - 1) otherwise
```

221. Implement this recurrence:

```
g(t,a,b) is t if b is zero
g(t,a,b) is g(t,a * a,b / 2) if b is even
g(t,a,b) is g(t * a,a,b - 1) otherwise
```

222. Implement this recurrence:

```
g(t,a,b) is b if a is the empty list
g(t,a,b) is a if b is the empty list
g(t,a,b) is [a[0]] + g(1,a[1:],b) if t is 0
g(t,a,b) is [b[0]] + g(0,a,b[1:]) otherwise
```

223. Implement this recurrence:

```
f(a) is g(a[0],a[1:])
g(b,c) is b if c is the empty list
g(b,c) is g(c[0],c[1:]) if c[0] < b
g(b,c) is g(b,c[1:]) otherwise
```

224. Implement this recurrence:

```
f(a) is g(a[0],a[1:])
g(b,c) is b if c is the empty list
g(b,c) is f(c) if c[0] > b
g(b,c) is g(b,c[1:]) otherwise
```

Major Concept: Loops

Concept: recognizing *counts* and *accumulations*

225. Which pattern does the following function implement?

```
def g(items):
    m = 0
    for i in range(0,len(items),1):
        m = m + items[i]
    return m
```

(a) the *filtered-counting* pattern
(b) the *counting* pattern
(c) the *accumulate* pattern
(d) the *filtered-accumulate* pattern

226. Which pattern does the following function implement?

```
def f(items):
    m = 0
    for i in range(0,len(items),1):
        if (isPrime(items[i])):
            m = m + items[i]
    return m
```

(a) the *accumulate* pattern
(b) the *filtered-accumulate* pattern
(c) the *filtered-counting* pattern
(d) the *counting* pattern

227. Which pattern does the following function implement?

```
def g(items):
    m = 0
    for i in range(0,len(items),1):
        m = m + 1
    return m
```

(a) the *filtered-counting* pattern
(b) the *accumulate* pattern
(c) the *counting* pattern
(d) the *filtered-accumulate* pattern

228. Which pattern does the following function implement?

```
def f(items):
    m = 0
    for i in range(0,len(items),1):
        if (isX(items[i])):
            m = m + 1
    return m
```

(a) the *counting* pattern
(b) the *accumulate* pattern
(c) the *filtered-counting* pattern
(d) the *filtered-accumulate* pattern

Concept: picking patterns, *counts* and *accumulate*

229. If I wish to solve this problem: *determine the sum of the numbers from* a *to* b, I should implement the:

(a) the *accumulate* pattern
(b) the *filtered-accumulate* pattern
(c) the *counting* pattern
(d) the *filtered-counting* pattern

230. If I wish to solve this problem: *count the number of letters in a string*, I should implement the:

(a) the *filtered-accumulate* pattern
(b) the *filtered-counting* pattern
(c) the *accumulate* pattern
(d) the *counting* pattern

231. If I wish to solve this problem: *count the number of Ramanujan numbers in the range* a *to* b, I should implement the:

(a) the *filtered-counting* pattern
(b) the *accumulate* pattern
(c) the *filtered-accumulate* pattern
(d) the *counting* pattern

232. If I wish to solve this problem: *determine the sum of the Ramanujan numbers in the range* a *to* b, I should implement the:

(a) the *filtered-accumulate* pattern
(b) the *filtered-counting* pattern
(c) the *accumulate* pattern
(d) the *counting* pattern

233. If I wish to solve this problem: *determine the product of the numbers in a list*, I should implement the:

(a) the *filtered-accumulate* pattern
(b) the *counting* pattern
(c) the *filtered-counting* pattern
(d) the *accumulate* pattern

234. If I wish to solve this problem: *determine the product of the odd numbers in a list*, I should implement the:

(a) the *accumulate* pattern
(b) the *filtered-counting* pattern
(c) the *counting* pattern
(d) the *filtered-accumulate* pattern

235. If I wish to solve this problem: *shuffle two lists, creating a third list*, I should implement the:

(a) the *shuffle* pattern

(b) the *shuffle-counting* pattern

(c) the *filtered-shuffle* pattern

(d) the *filtered-accumulate* pattern

Concept: *recognizing map, filter, search, extreme, and extreme-index*

236. Which pattern does the following function implement?

```
def g(f,s):
    u = []
    for j in range(0,len(s),1):
        u = u + [f(s[j])]
    return u
```

(a) the *extreme* pattern

(b) the *extreme index* pattern

(c) the *map* pattern

(d) the *search* pattern

(e) the *filter* pattern

237. Which pattern does the following function implement?

```
def h(g,t):
    for j in range(0,len(t),1):
        if (t[j] == g)
            return True
    return False
```

(a) the *search* pattern

(b) the *map* pattern

(c) the *extreme* pattern

(d) the *extreme index* pattern

(e) the *filter* pattern

238. Which pattern does the following function implement?

```
def f(h,r):
    w = False
    for i in range(0,len(r),1):
        if (r[i] == h)
            w = True
    return w
```

(a) the *extreme* pattern

(b) the *search* pattern

(c) the *filter* pattern

(d) the *map* pattern

(e) the *extreme index* pattern

239. Which pattern does the following function implement?

```
def h(isG,t):
    v = []
    for j in range(0,len(t),1):
        if (isG(t[j])):
            v = v + [t[j]]
    return v
```

(a) the *extreme* pattern

(b) the *extreme index* pattern

(c) the *map* pattern

(d) the *search* pattern

(e) the *filter* pattern

240. Which pattern does the following function implement?

```
def f(s):
    u = []
    for j in range(0,len(s),1):
        if (isX(s[j])):
            u = u + [s[j]]
    return u
```

(a) the *filter* pattern

(b) the *map* pattern

(c) the *extreme index* pattern

(d) the *search* pattern

(e) the *extreme* pattern

241. Which pattern does the following function implement?

```
def f(s):
    u = s[0]
    for j in range(1,len(s),1):
        if (s[j] > u):
            u = s[j]
    return u
```

(a) the *extreme* pattern

(b) the *map* pattern

(c) the *extreme index* pattern

(d) the *filter* pattern

(e) the *search* pattern

242. Which pattern does the following function implement?

```
def h(r):
    w = 0
    for j in range(1,len(r),1):
        if (r[j] > r[w]):
            w = j
    return r[w]
```

(a) the *extreme* pattern

(b) the *map* pattern

(c) the *search* pattern

(d) the *extreme index* pattern

(e) the *filter* pattern

243. Which pattern does the following function implement?

```
def h(r):
    w = 0
    for i in range(1,len(r),1):
        if (r[i] < r[w]):
            w = i
    return w
```

(a) the *extreme* pattern

(b) the *map* pattern

(c) the *search* pattern

(d) the *extreme index* pattern

(e) the *filter* pattern

Concept: *picking patterns – map, filter, search, extreme, and extreme-index*

244. If I wish to solve this problem: *square every number in a list*, I should implement the:

(a) the *filter* pattern

(b) the *search* pattern

(c) the *accumulate* pattern

(d) the *map* pattern

245. If I wish to solve this problem: *find if there is an even number in a list*, I should implement the:

(a) the *map* pattern

(b) the *filter* pattern

(c) the *search* pattern

(d) the *accumulate* pattern

246. If I wish to solve this problem: *extract the prime numbers from a list*, I should implement the:

(a) the *map* pattern

(b) the *filter* pattern

(c) the *search* pattern

Concept: *chaining patterns*

247. If I wish to solve this problem: *sum the squares of every number in a list*, I should implement the:

(a) the *filter* pattern followed by the *accumulate* pattern

(b) the *map* pattern followed by the *accumulate* pattern

(c) the *map* pattern followed by the *search* pattern

(d) the *filter* pattern followed by the *search* pattern

(e) the *count* pattern followed by the *accumulate* pattern

248. If I wish to solve this problem: *find the largest even number in a list*, I should implement the:

(a) the *filter* pattern followed by the *search* pattern

(b) the *map* pattern followed by the *accumulate* pattern

(c) the *map* pattern followed by the *search* pattern

(d) the *count* pattern followed by the *accumulate* pattern

(e) the *filter* pattern followed by the *extreme* pattern

249. If I wish to solve this problem: *extract the even numbers from a list and then square each item in the resulting list*, I should implement the:

(a) the *map* pattern followed by the *accumulate* pattern

(b) the *filter* pattern followed by the *map* pattern

(c) the *map* pattern followed by the *filter* pattern

(d) the *filter* pattern followed by the *extreme* pattern

(e) the *count* pattern followed by the *accumulate* pattern

250. If I wish to solve this problem: *increment each number in a list and then extract the prime numbers*, I should implement the:

(a) the *map* pattern followed by the *filter* pattern

(b) the *count* pattern followed by the *accumulate* pattern

(c) the *filter* pattern followed by the *map* pattern

(d) the *filter* pattern followed by the *extreme* pattern

(e) the *map* pattern followed by the *accumulate* pattern

Concept: *verifying code*

251. Consider the problem statement: *sum the numbers from* a *to* b (*inclusive*). Does this function compute the correct result?

```
def sum(a,b):
    total = 0:
    for i in range(0,b,1):
        total = total + a
    return total
```

(a) No

(b) Yes

252. Consider the problem statement: *sum the numbers from* a *to* b (*inclusive*). Does this function compute the correct result?

```
def sum(a,b):
    total = 0:
    for i in range(a,b,1):
        total = total + i
    return total
```

(a) Yes

(b) No

253. Consider the problem statement: *sum the numbers from* a *to* b (*inclusive*). Does this function compute the correct result?

```
def sum(a,b):
    total = a
    for i in range(a+1,b+1,1):
        total = total + i
    return total
```

(a) Yes

(b) No

254. Consider the problem statement: *sum the numbers from* a *to* b (*inclusive*). Does this function compute the correct result?

```
def sum(a,b):
    total = b
    for i in range(a,b,0):
        total = total + i
    return total
```

(a) No

(b) Yes

255. Consider the problem statement: *sum the numbers from* a *to* b (*inclusive*). Does this function compute the correct result?

```
def sum(a,b):
    total = a
    for i in range(a,b+1,1):
        total = total + i
    return total
```

(a) Yes

(b) No

256. Consider the problem statement: *sum the numbers from* a *to* b (*inclusive*). Does this function compute the correct result?

```
def sum(a,b):
    total = 0
    for i in range(a,b+1,1):
        total = total + i
    return total
```

(a) Yes

(b) No

257. Consider the problem statement: *shuffle two lists, creating a third list.* Does this function compute the correct result if the lists are of *unequal* lengths?

```
def shuffle(list1,list2):
    list3 = []
    for j in range(0,len(list2),1):
        list3 = list3 + [list2[j],list1[j]]
    return list3
```

(a) No, the function always runs without error, but may not correctly shuffle

(b) No, the function can fail with an error

(c) Yes

258. Consider the problem statement: *shuffle two lists, creating a third list.* Does this function compute the correct result if the lists are of *unequal* lengths?

```
def shuffle(list1,list2):
    list3 = []
    j = 0; i = 0
    while (j < len(list1) and i < len(list2)):
        list3 = list3 + [list1[j],list2[i]]
        j += 1; i += 1

    return list3 + list1[j:] + list2[i:]
```

(a) No, the function can fail with an error

(b) No, the function always runs without error, but may not correctly shuffle

(c) Yes

259. Consider the problem statement: *shuffle two lists, creating a third list.* Does this function compute the correct result if the lists are of *unequal* lengths?

```
def shuffle(list1,list2):
    list3 = []
    j = 0; i = 0
    while (j < len(list1) and i < len(list2)):
        list3 = list3 + [list1[j],list2[i]]
        j += 1; i += 1

    return list3
```

(a) Yes

(b) No, the function can fail with an error

(c) No, the function always runs without error, but may not correctly shuffle

260. Consider the problem statement: *shuffle two lists, creating a third list.* Does this function compute the correct result if the lists are of *unequal* lengths?

```
def shuffle(list1,list2):
    list3 = []
    n = 0; j = 0; i = 0
    while (j < len(list1) and i < len(list2)):
        if (n % 2 == 0):
            list3 = list3 + [list1[j]]
            j += 1
        else:
            list3 = list3 + [list2[i]]
            i += 1
        n += 1

    return list3
```

(a) No, the function always runs without error, but may not correctly shuffle

(b) No, the function can fail with an error

(c) Yes

261. Consider the problem statement: *shuffle two lists, creating a third list.* Does this function compute the correct result if the lists are of *unequal* lengths?

```
def shuffle(list1,list2):
    list3 = []
    n = 0; j = 0; i = 0
    while (j < len(list1) and i < len(list2)):
```

```
        if (n % 2 == 0):
            list3 = list3 + [list1[j]]
            j += 1
        else:
            list3 = list3 + [list2[i]]
            i += 1
        n += 1

    return list3 + list1[j:] + list2[i:]
```

(a) No, the function can fail with an error

(b) No, the function always runs without error, but may not correctly shuffle

(c) Yes

262. Consider the problem statement: *shuffle two lists, creating a third list.* Does this function compute the correct result if the lists are of *unequal* lengths?

```
def shuffle(list1,list2):
    list3 = []
    n = 0; i = 0
    for j in range(0,len(list1),1):
        if (n % 2 == 0):
            list3 = list3 + [list1[j]]
        elif (i < len(list2)):
            list3 = list3 + [list2[i]]
            i += 1
        n += 1

    return list3 + list2[i:]
```

(a) No, the function always runs without error, but may not correctly shuffle

(b) No, the function can fail with an error

(c) Yes

263. Consider the problem statement: *shuffle two lists, creating a third list.* Does this function compute the correct result if *list1* is longer than *list2*?

```
def shuffle(list1,list2):
    list3 = []
    m = 0; j = 0
    for i in range(0,len(list1),1):
        if (m % 2 == 0):
            list3 = list3 + [list1[i]]
        else:
            list3 = list3 + [list2[j]]
            j += 1
        m += 1

    return list3
```

(a) No, the function always runs without error, but may not correctly shuffle

(b) No, the function can fail with an error

(c) Yes

Concept: *implementing loops*

264. Implement this loop: total up the product of the numbers from 1 to x, inclusive.

265. Implement this loop: total up the product of the numbers from a to b, inclusive.

266. Implement this loop: total up the sum of the numbers from a to b, inclusive.

267. Implement this loop: total up the sum of the numbers from 1 to x, inclusive.

268. Implement this loop: count the number of characters in a string s.

269. Implement this loop: count the number of uppercase characters in a string s.
Here is an implementation of an *isUpper* function, if you want to test your loop:

```
def isUpper(x): return x.isupper()
```

270. Implement this loop: count the number of vowels in a string s.
Here is an implementation of an *isVowel* function, if you want to test your loop:

```
def isVowel(ch): return ch in "aAeEiIoOuU"
```

271. Implement this loop: determine a^b by updating an accumulator via multiplication by a, b times.

272. Implement this loop: count the number of prime numbers from a to b inclusive.
Here is a naive implementation of an *isPrime* function, if you want to test your loop:

```
def isPrime(n):
    for i in range(2,int(pow(n,0.5)) + 2):
        if (n % i == 0):
            return False
    return True
```

273. Implement this loop: count the number of numbers that are divisible by 2 or 3 from a to b inclusive.

Concept: *loop conversions*

274. Convert this *while* loop to a *for* loop:

```
i = 0
while (i < x + 1):
    print(x)
    i = i + 1
```

275. Convert this *while* loop to a *for* loop:

```
i = 2
while (i < x):
    print(x)
    i = i + 1
```

276. Convert this *while* loop to a *for* loop:

```
i = 3
while (i < x):
    print(x)
    i = i + 2
```

277. Convert this *while* loop to a *for* loop:

```
i = 0
while (i < len(s)):
    print(s[i])
    i = i + 1
```

278. Convert this *for* loop to a *while* loop:

```
for i in range(2,len(s),2):
    print(s[i])
```

279. Convert this *for* loop to a *while* loop:

```
count = 0
for i in range(0,len(s),2):
    if (s.upper(i))
        count += 1
```