(1-6) When the following code is executed in the Python Shell,

```python
>>> mypi
??-1
>>> mypi = 3
>>> mypi
??-2
>>> w = mypi
>>> id(w) == id(mypi)
??-3
>>> type(w)
??-4
>>> mypi = [3]
>>> id(w) == id(mypi)
??-5
```

(1) Replace ??-1 with the correct value:

a) 3  
 b) [1, 2, 3, 4]  
 c) True  
 d) False  
 e) NameError

(2) Replace ??-2 with the correct value:

a) 3  
 b) [1, 2, 3, 4]  
 c) True  
 d) False  
 e) NameError

(3) Replace ??-3 with the correct value:

a) 3  
 b) [1, 2, 3, 4]  
 c) True  
 d) False  
 e) NameError

(4) Replace ??-4 with the correct value:

a) int  
 b) float  
 c) str  
 d) boolean  
 e) tuple

(5) Replace ??-5 with the correct value:

a) 3  
 b) [1, 2, 3, 4]  
 c) True  
 d) False  
 e) NameError

(6) Which statement is correct?

a) mypi is defined in the **global** namespace; w is defined in the **global** namespace.

b) mypi is defined in the **global** namespace; w is defined in the **local** namespace.

c) mypi is defined in the **local** namespace; w is defined in the **global** namespace.

d) mypi is defined in the **local** namespace; w is defined in the **local** namespace.

e) mypi is defined in the **local** namespace; w is no longer defined.
Given function `approx_sqrt`:

```python
def approx_sqrt(num, iterations):
    '''(number, int) -> float
    Generates an approximate square root of num, a positive integer, via an iterative process that runs iterations times. The approximate square root is returned.
    >>> approx_sqrt(1, 1)
    1.0
    >>> approx_sqrt(4, 1)
    ?-1
    >>> approx_sqrt(4, 5)
    2.000000000000002
    '''
    value = 1
    for ctr in range(iterations):
        value = .5 * (value + num/value)
    return ??-2
```

(7) Replace `??-1` with the correct code:

a) 4  
b) 1  
c) 2  
d) 2.0  
e) 2.5

(8) Replace `??-2` with the correct code:

a) num  
b) value  
c) 2  
d) 2.0  
e) 2.5

(9) When `>> approx_sqrt(4, 5)` is executed, the last value of `ctr` is

a) 0  
b) 1  
c) 4  
d) 5  
e) 2.000000000000002

(10) `approx_sqrt` is an example of

a) recursion  
b) Monte Carlo algorithm  
c) accumulator pattern  
d) encryption algorithm  
e) REPL
(11-13) Given the following Python code:

```python
def dtobr(n):
    '''(int) -> str
    Use recursive algorithm to convert n >= 0 to a binary string.
    '''
    if n < 2:
        return str(n)
    else:
        return dtobr(n // 2) + str(n % 2)
```

When `>>> dtobr(6)` is executed,

(11) `dtobr` is called ?? times (including the first time):

a) 0  b) 1  c) 2  d) 3  e) 4

(12) The last time `dtobr` is called, the value of `n` is

a) 0  b) 1  c) 2  d) '0'  e) '1'

(13) The best set of test cases for `dtobr` is

a)
<table>
<thead>
<tr>
<th>dtobr(6)</th>
<th>dtobr(2)</th>
<th>dtobr(4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>dtobr(0)</td>
<td>dtobr(4)</td>
<td>dtobr(0)</td>
</tr>
<tr>
<td>dtobr(1)</td>
<td>dtobr(6)</td>
<td>dtobr(1)</td>
</tr>
<tr>
<td>dtobr(2)</td>
<td>dtobr(24)</td>
<td>dtobr(15)</td>
</tr>
<tr>
<td>dtobr(17)</td>
<td>dtobr(36)</td>
<td>dtobr(31)</td>
</tr>
<tr>
<td>dtobr(50)</td>
<td>dtobr(44)</td>
<td>dtobr(63)</td>
</tr>
</tbody>
</table>

(14) Given the following UNTESTED Python code:

```python
def charCt(s, c):
    '''(str, str) -> int
    Return count of occurrences of char c in string s.
    '''

    ct = 0
    for ch in s:
        if ch == c:
            ct += 1
    return ct
```

```python
>>> charCt('hello, world', 'o')
2
```
The set of test cases that will find the bug in `charC` is

a) `charC('','a')`  
b) `charC('abc','a')`  
c) `charC('abc','a')`  
charC('a','a')  
charC('a','a')  
charC('abc','a')  
charC('abc','a')  
charC('ghi','x')  
charC('x','x')

(15-17) After the following Python code is executed:

```python
>>> x = [99.9]
>>> y = x
>>> z = [99.9]
>>> y[0] = 0
```

(15) The value of `x` is

a) 99.9  
b) [99.9]  
c) 0  
d) [0]  
e) None

(16) The value of `y` is

a) 99.9  
b) [99.9]  
c) 0  
d) [0]  
e) None

(17) The value of `z` is

a) 99.9  
b) [99.9]  
c) 0  
d) [0]  
e) None

(18-22) Given the following Python code:

```python
def taxable(inc, exempt, STD_E, STD_D):
    '''(number, int, number, number)
    Adjust gross income (inc) to taxable income by applying standard deduction and exemptions.
    CALLED BY: est_tax
    >>> taxable(20000, 1, 4150, 6500)
    9350
    '''
    #print(income)
    #print(salary)
taxable_income = inc - STD_D
exempt_adjust = STD_E * exempt
taxable_income = taxable_income - exempt_adjust
return(taxable_income)
```
def est_tax(income, exemptions):
    '''(number, int) \rightarrow None

    Generates an estimate for federal income tax. CALLS: taxable

    >>> est_tax(20000, 1)
    1870.0
    '''

    STD_EXEMPT = 4150
    STD_DEDUCT = 6500
    TAX_RATE = .20

    taxable_income = taxable(income, exemptions, STD_EXEMPT, STD_DEDUCT)
    estimated_tax = taxable_income * TAX_RATE

    #print('Estimated tax is:', estimated_tax)
    return None

1 def main(salary, exemptions):
    2     '''driver for estimated tax functions'''
    3     result = est_tax(salary, exemptions)
    4     print(result)
    5     print(salary)
    6     print(taxable_income)
    7     return None

salary = 20000
exemptions = 1
main(salary, exemptions)

(18) After line 4 in main is executed, what will be printed?

a) 1870.0   b) 20000   c) None   d) NameError

(19) After line 5 in main is executed, what will be printed?

a) 1870.0   b) 20000   c) None   d) NameError

(20) After line 6 in main is executed, what will be printed?

a) 1870.0   b) 20000   c) None   d) NameError

(21) If the #print(income) line of code in taxable were executed, what would be printed?

a) 1870.0   b) 20000   c) None   d) NameError

(22) If the #print(salary) line of code in taxable were executed, what would be printed?

a) 1870.0   b) 20000   c) None   d) NameError
(23) Given the following Python code:

```python
def q23(s):
    """midterm function""
    if len(s) == 1:
        return True
    elif len(s) == 2:
        return s[0] == s[1]
    elif s[0] != s[-1]:
        return False
    else:
        return q23(s[1:])

>>> q23('abcdeffedcba')
```

The second time `q23` is called, the value of `s` is

a) True  b) False  c) 'abcdeffedcba'  d) 'bcdeffedcb'  c) 'abcddcba'

(24-28) Replace the ??s with the results of executing the following code in the Python Shell.

```python
>>> states = [2, 'OR', 'WA']
>>> states.append('ID')
>>> states[0] += 1
>>> states
??-1
>>> state = 'mt'
>>> state.upper()
??-2
>>> states.append(state)
>>> states[0] += 1
>>> states
??-3
>>> pnw = states.copy()
>>> states = states.append('CA')
>>> states[0] += 1
??-4
>>> pnw
??-5
```
(24) Replace ??–1 with the correct result:
a) [2, 'OR', 'WA'] b) [3, 'OR', 'WA', 'ID'] c) [3, 'ID', 'OR', 'WA']
d) None e) NameError

(25) Replace ??–2 with the correct result:
 a) 'mt' b) 'MT' c) None d) TypeError e) NameError

(26) Replace ??–3 with the correct result:
 a) [4, 'OR', 'WA', 'ID', 'mt'] b) [4, 'OR', 'WA', 'ID', 'MT']
 c) None d) TypeError e) NameError

(27) Replace ??–4 with the correct result:
 a) [4, 'OR', 'WA', 'ID', 'mt'] b) [4, 'OR', 'WA', 'ID', 'MT']
 c) None d) TypeError e) NameError

(28) Replace ??–5 with the correct result:
 a) [4, 'OR', 'WA', 'ID', 'mt'] b) [4, 'OR', 'WA', 'ID', 'MT']
 c) None d) TypeError e) NameError

(29) TypeError, NameError, ZeroDivisionError are examples of which type of error?
 a) syntax b) runtime c) logical/semantic d) regression e) integrated

(30) The error that occurs when >>> charCt('hello, world', 'o') (question 14) is executed is an example of which type of error?
 a) syntax b) runtime c) logical/semantic d) regression e) integrated

(31) The decimal representation of binary 11111 is:
 a) 11, 111 b) 63 c) 64 d) 31 e) 32
(32-34) Given the following Python code:

```python
roman = {'I': 1, 'V': 5, 'X': 10, 'L': 50, 'C': 100, 'M': 1000}
```

What is the result of executing

(32) ```>>> roman['X']```  

a) True  
b) False  
c) 10  
d) 'X'  
e) TypeError  

(33) ```>>> 'V' in roman```  

a) True  
b) False  
c) 5  
d) 'X'  
e) TypeError  

(34) ```>>> 5 in roman```  

a) True  
b) False  
c) 'V'  
d) 'X'  
e) TypeError  

(35) Given the following Python code:

```python
1  def drawShape(s):
2      '''(int) --&gt; None
3      Draw a square with sides of length s.
4  
5      >>> drawShape(100)
6      [draws a square with sides length 100]
7      '''
8  
9      turn = 90
10     for i in range(4):
11         fd(s)
12         lt(turn)
13     return None
```

Which lines of code would need to be changed to revise `drawShape` to draw an n-sided polygon, where n is a new argument to the function?  

a) 1, 2, 14  
b) 1, 2, 4, 6, 7, 9, 10  
c) 1, 2, 4, 6, 7, 11, 12  
d) 1, 2, 4, 6, 7, 14  
e) 1, 6, 14
(36-39) Given the following Python code:

```python
def greeting(f, s):
    '''Type contract goes here. '''
    f(s)
    return None

def hello(s):
    '''Type contract goes here. '''
    print('Hello, ' + s + '.
    return None

def ciao(s):
    '''Type contract goes here. '''
    greeting = 'Ciao, ' + s + '.
    print(greeting)
    return greeting
```

(36) The type contract for function `greeting` is

a) `(str, str) -> None`  b) `(str, str) -> str`  c) `(function, str) -> str`

d) `(function, str) -> None`  e) `(str, str) -> boolean`

(37) The type contract for function `ciao` is

a) `(str) -> None`  b) `(str) -> str`  c) `(function) -> str`

d) `(function) -> None`  e) `(function) -> function`

(38) What is the result when the following code is executed?

```python
>>> greeting(hello, 'World')
```

a) 'Hello, World.' is returned  b) Hello, World. is printed
c) 'Hello, ' + s + ' . is printed  d) None is printed

(39) The result of executing function `hello` is an example of a(n)

a) recursion  b) Monte Carlo algorithm  c) accumulator pattern  d) side effect
(40) What is the result when the following Python code is executed?

```python
te = 'End of multiple choice.'
print('{}'.format(te))
print(f'{te}')
print('{} % (te))
```

a)
End of multiple choice.
End of multiple choice.
End of multiple choice.
(41-43)

(41) (8 pts.) Write an iterative function, add_digits, to sum the digits of an integer, n, where 100 <= n <= 999. All variables should be of type integer (no strings, lists, etc.). The sum is printed; add_digits should return None.

The function should be written using CIS 210 style guidelines. The docstring examples of use should include at least one example with a “normal” value argument and at least one example with a “boundary” value argument. Python code should clearly reflect the underlying algorithm. Code should use only the most appropriate Python “tools” for solving the problem.
(42) (6 pts.) Write an iterative function, `add_more_digits`, to sum the digits of a non-negative integer, `n`. The sum is returned. All variables should be of type integer (no strings, lists, etc.).

The function should be written using CIS 210 style guidelines, except that the docstring examples of use may be omitted. Python code should clearly reflect the underlying algorithm. Code should use the most appropriate Python “tools” for solving the problem.

(43) (6 pts.) Write a function, `add_more_digits_r`, which implements a recursive algorithm to sum the digits of a non-negative integer, `n`. All variables should be of type integer (no strings, lists, etc.). The sum is returned.

The function should be written using CIS 210 style guidelines, except that the docstring examples of use may be omitted. Python code should clearly reflect the underlying algorithm. Code should use the most appropriate Python “tools” for solving the problem.