Section 1

Quiz 1

Question 1: Which of these is the correct way to define (assign a value to) a variable in Python?

A. x:25
B. x = 25
C. x(25)
D. print(25)

Answer B

Question 2: What would the output be if you run the code below?

```
1 | x = 25
2 | print(x)
3 | x = 30
4 | print(x)
```

A. 25

   30

B. 25

   25

C. This would give you an error because x already has a value when we do x = 30.

Answer A

Question 3: Which of these is not a valid Python variable name?

A. myVariable
B. my_variable
C. _var
D. 1st_variable

Answer D

Question 4: Why would you write a variable name in all uppercase (all capital letters) in Python?

A. There's no reason to ever do this in Python. It's the wrong style.
B. To signal that you don't intend for this variable's value to change.
C. When defining important values that will be used throughout the entire code.

Answer B

Quiz 2

Question 1: What would the value of x be at the end of this code segment?

```
a = 17.5
b = 5
x = a + b - 9
```

A. 14
B. 13.5
C. 14.0
D. x has no value
E. This would result in an error.

Answer: B

Question 2: What would the output be (i.e., what would the user see) from this code sample?

```
friends = 9
near = 3
friends_away = friends - near
```

A. The user would see nothing.
B. 6
C. This would result in an error, because you can't use underscores in variable names.

Answer: A

Question 3: What would the value of div be at the end of this segment?

```
my_variable = 7
div = my_variable / 2
```

A. 4
B. 3.5
C. 3

Answer: B

Question 4:

Which of these two is correct in Python 3? (That is the version the videos are using!)

A. print('Hello, world!')
B. print 'Hello, world!'

Answer: A

Question 5: What would the user see as output after this code segment?

```
row_number = 235
result = row_number % 2
print(result)
```

A. 1
B. The user would see an error.
C. 117.5

Answer: A

Question 6: What would the value of x be at the end of this code segment?

```
x = 15 // 2
```

A. 7.5
B. 7
C. 8

Answer: B

Quiz 3

Question 1: What would the user see after running this code segment?

```
name = 'Jose'
greeting = 'Hello, ' + name
print(greeting)
```

A. Hello, Jose
B. 'Hello, Jose'
C. "Hello, Jose"
D. The user would see nothing, or there would be an error.

Answer: A

Question 2: What would the user see if they executed this segment of code?

```
another_greeting = f'How are you, {name}?
print(another_greeting)
```

A. How are you, Jose?
B. 'How are you, Jose?'
C. They would see an error.
D. They would see nothing printed out.

Answer: C

Question 3: What would the output of this code segment be?

```
name = 'Jose'
friend = 'Rolf'
phrase = name + ' is friends with ' + friend
print(phrase)
```

A. Jose is friends with Rolf
B. Jose is friends with Rolf.
C. Rolf is friends with Jose
D. They would see an error, because you cannot add strings together.

Answer: A

Question 4: What would you expect the output to be when running this code?

```
name = "Rolf Smith"
street = "123 No Name Road"
postcode = "PY10 1CP"

address = f"""Name: {name}
Street: {street}
Postcode: {postcode}
Country: United Kingdom"""

print(address)
```

A. ""Name: Rolf Smith

Street: 123 No Name Road

Postcode: PY10 1CP

Country: United Kingdom""

B. Name: Rolf Smith

Street: 123 No Name Road

Postcode: PY10 1CP

Country: United Kingdom

C. There would be an error because you can't use f-strings with multi-line strings.

Answer: B

Question 5: What would the output of this code be?

```
description = "{} is {} years old."
print(description.format("Bob", 30))
```

A. There would be an error, because we've tried to put a number into a string.
B. There would be an error, because we've got multiple placeholders.

C. Bob is Bob years old.
D. Bob is Bob years old.

   30 is 30 years old.

E. Bob is 30 years old.

Answer: D

Question 6: Would this be valid?

```
description = "{} is {age} years old."
print(description.format("Bob", age=30))
```

A. This is not valid because we can only use names within curly braces when using f-strings.
B. This is not valid because all sets of curly braces must have names, and here one has a name but one does not.
C. Yes, this would be OK!

Answer: C

Quiz 4

Question 1:

What would the user see when running this code?

Assume the user enters `Jose` as their name.

```
your_name = input('Enter your name: ')
print(f'Hello {your_name}!')
```

A. Enter your name: Jose

   Hello Jose!

B. Enter your name:
C. Hello Jose!

Answer: A

Question 2:

What would the value of double be after running this code segment?

Assume the user enters `3` when asked to enter something.

```
user_input = input('Please enter a number: ')
double = user_input * 2
print(f'Your number doubled is {double}.')
```

A. '33'
B. '6'
C. Nothing. The user would get an error.
D. Your number doubled is 6.

Answer: A

Question 3:

What would the value of `z` be at the end of this segment?

```
x = '56'
z = int(x) * 2
```

A. '5656'
B. 112
C. Nothing—there would be an error.

Answer: B

Quiz 5

Question 1: Which of the following is not a valid Python keyword?

A. true
B. True
C. False

Answer: A

Question 2: What would the result of these operations be?

```
15 < 20
20 < 20
21 < 20
97 == 97
100 >= 100
100 > 100
```

a.

```
15 < 20  # True
20 < 20  # False
21 < 20  # False
97 == 97  # True
100 >= 100  # True
100 > 100  # False
```

b.

```
15 < 20  # False
20 < 20  # False
21 < 20  # False
97 == 97  # True
100 >= 100  # False
100 > 100  # False
```

c.

```
15 < 20  # True
20 < 20  # True
21 < 20  # False
97 == 97  # True
100 >= 100  # True
100 > 100  # False
```

Answer: A

Question 3: What would the result of cmp be after running this code segment?

```
cmp = 15 > 20 or 17 < 20
```

A. True
B. False
C. There would be an error because you can't use multiple comparisons on one line.

Answer: A

Question 4: What does or do?

A. It returns True if either value is True.
B. It returns True only if both values are True.
C. It returns the value after the or, if the value before the or evaluates to False.

Answer: C

Question 5: What would the value of cmp be at the end of this code segment?

```
x = True
cmp = x and 18
```

A. It would be an error; you cannot have 18 after an and.
B. 18
C. True
D. False

Answer: B

Question 6:

What would the output of this code be? Imagine the user enters 16.

```
age = int(input("Enter your age: "))
side_job = True
print(age > 18 and age < 65 or side_job)
```

A. True
B. False
C. This would be an error because we've use 'and' and 'or' in the same line, and Python might be confused as to the order of operations.

Answer: A

Quiz 6

Question 1: Which one is a difference between a list and a tuple?

A. Lists are a collection of elements, whereas tuples are not.
B. You can add elements to a list, you cannot add them to a tuple.
C. Lists are ordered, tuples are not.

Answer: B

Question 2: You can have duplicate elements in a set.

A. True
B. False

Answer: B

Question 3: Which of the following is not a tuple?

```
A. t = (15)
B. t = (15,)
C. t = 15,
```

Answer: A

Question 4: How could you make this list longer?

```
1  my_list = ['13', '45']
```

```
A. my_list.add('56')
B. my_list.append('56')
C. my_list.push('56')
```

Answer: B

Question 5: With this code below:

```
1  friends = {"Charlie", "Jen", "Anne"}
```

How would you add `"Rolf"` to the `friends` set?

A. friends.append("Rolf")
B. friends.add("Rolf")
C. You can't add to a set after creating it.

Answer: B

Question 6: What would `set_three` contain at the end of this code segment?

```
set_one = {1, 2, 3, 4, 5}
set_two = {1, 3, 5, 7, 9, 11}
set_three = set_one.intersection(set_two)
```

A. {1, 3, 5}
B. {1, 2, 3, 4, 5}
C. This would give an error.
D. [1, 3, 5]

Answer: A

Quiz 7

Question 1: Which of the following best describes a dictionary?

A. A set of key-value pairs. All keys must be unique.
B. A list of key-value pairs.
C. A collection of key-value pairs, where you can have duplicate keys.

Answer: A

Question 2: Given the following dictionary:

```
my_friends = {
    'Jose': {'last_seen': 6},
    'Rolf': {'surname': 'Smith'},
    'Anne': 6
}
```

What would the value of this code be?

```
my_friends['Jose']
```

A. 6
B. 'last_seen'
C. {'last_seen': 6}
D. This would give you an error.

Answer: C

Question 3: Given the following code:

```
my_friends = {
    'Jose': {'last_seen': 6},
    'Rolf': {'surname': 'Smith'},
    'Anne': 6
}
```

What would the value of this code be?

```
my_friends['Jose']['last_seen']
```

A. This would give you an error, because you're accessing the 'last_seen' property of ['Jose'], which you cannot do.
B. 6
C. This would give you an error for some reason other than what the other answer suggests.

Answer: B

Question 4:

What would the output of the following code be?

```
players = [
    {
        'name': 'Rolf',
        'numbers': (13, 22, 3, 6, 9)
    },
    {
        'name': 'John',
        'numbers': (22, 3, 5, 7, 9)
    }
]
print(players[0]['numbers'])
print(players[0]['numbers'][0])
```

A. This would give an error, because we're accessing the key `0` of a dictionary, but it doesn't have such key.
B. This would give an error, because you cannot have something like `players[0]['numbers'][0]`
C.
```
(13, 22, 3, 6, 9)
13
```
D. It's difficult to say, because the numbers in the tuple will be in random order when they are printed out.

Answer: C

Section 2

Quiz 8

Question 1: How would you check if Rolf is older than Dane?

```
rolf = 35
dane = 27
```

A.
```
if rolf>=dane:
  print("Rolf is older")
```

B.
```
else rolf > dane:
  print("Rolf is older")
```

C.
```
if rolf>>dane:
  print("Rolf is older")
```

D.

```
if rolf > dane:
    print("Rolf is older")
```

Answer: D

Question 2: Replace the ??? line with one of the answers to complete an example of password authentication:

```
PASSWORD = "secret1"

???

if user_input == PASSWORD:
    print("Welcome back.")
else:
    print("Wrong password.")
```

A. `user_input = input("Password: ")`
B. `input("Password")`
C. `user_input = input`
D. `user_input: input("Password: ")`

Answer: A

Question 3: The below is not working Python code, but it represents a pattern of what we could do in Python (some call it pseudocode). Can you fill in the missing word, represented by _____?

```
if <condition> is True:
    <block to run if above condition is True>
____ <condition> is True:
    <block to run if above condition is True>
else:
    <block to run if none of the other conditions are True>
```

A. else if
B. elif
C. else

Answer: B

Quiz 9

Question 1: A while loop allows us to repeat a block of code for as long as a condition is True.

A. True.
B. False.

Answer: A

Question 2: Is this valid Python code for a while loop?

```
is_learning = True
while is_learning:
    print('You are awesome!')
```

A. Yes, although it will repeat many, many times.
B. No, this will give an error.

Answer: A

Question 3: Which of the following loops runs exactly 10 times?

A.
```
i = 0
while i <= 10:
    print(f'Repeated {i} times.')
    i += 1
```

B.
```
i = 0
while i < 10:
    print(f'Repeated {i} times.')
    i += 1
```

C.
```
i = 0
while i > 10:
    print(f'Repeated {i} times.')
    i += 1
```

Answer: B

Question 4: A for loop is for repeating things a specific number of times.

A. True.
B. False. It can be used for that, but it can also do other things.

Answer: B

Question 5: How would you iterate over a dictionary's keys and values?

Imagine you've got this dictionary defined:

```
my_friends = {
    'Jose': 6,
    'Rolf': 12,
    'Anne': 6
}
```

A.
```
for name, days in my_friends:
    print(f'I last saw {name} {days} days ago.')
```

B.
```
for name. days in my_friends.items():
    print(f'I last saw {name} {days} days ago.')
```

C.
```
for name, days in my_friends.items():
    print(f'I last saw {name} {days} days ago.')
```

Answer: C

Question 6:

Note: before answering, pay close attention to punctuation and symbols!

What would the output of this code be?

```
my_friends = {
    'Jose': 6,
    'Rolf': 12,
    'Anne': 6
}

do_i_know = 'Anne'

if do_i_know in my_friends:
  print(f'I know {do_i_know}')
```

A. I know Anne
B. I know Anne.
C. Nothing would get printed out.

Answer: A

Quiz 10

Question 1: Given this code, what would the value of `doubled` be at the end of the segment?

```
numbers = list(range(10))
doubled = [n*2 for n in numbers]
```

A. [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
B. [0, 2, 4, 6, 8]
C. [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
D. [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]

Answer: D

Question 2: A bit of a twist on the previous question! Given this code, what would the value of `numbers` be at the end of the segment?

```
numbers = list(range(10))
doubled = [n*2 for n in numbers]
```

A. [0, 1, 2, 3, 4, 5, 6 ,7, 8, 9]
B. [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]

C. [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
D. It would no longer exist.

Answer: A

Question 3: Given this list:

```
guests = ['Jose', 'Rolf', 'ruth', 'Charlie', 'michael']
```

How would you create a new list where all the names are uppercase?

(You may have to do some research as to what the correct way of turning a string into uppercase is!)

A. [name.lower() for name in guests]
B. [name.uppercase() for name in guests]
C. [name.upper() for name in guests]
D. [NAME for name in guests]

Answer: C

Question 4: Bonus question!

What would be the best way to turn this list of tuples into a dictionary?

```
guests = [('rolf', 25), ('adam', 28), ('jen', 24)]
```

A. {guest: age for guest, age in guests.items()}
B. dict(guests)
C. [guest, age for e in guests]
D. dict([guest, age for e in guests])

Answer: B

Quiz 11

Question 1: What is a function?

A. A series of instructions that execute top to bottom.
B. A block of code that you can run at any point after it has been defined, and that produces an output or performs an action. Sometimes it can take inputs as well.
C. Something that takes inputs and gives outputs.

Answer: B

Question 2: What is the correct syntax for defining a function in Python?

A. def name[]

   <block>

B. def name():

   <block>

C. def name:

<block>

Answer: B

Question 3: What would the output of this segment of code be?

```
def greet():
    print('Hey there Rolf')
```

A. Hey there Rolf
B. Nothing would be printed out.
C. This would give an error, because the function was not called.

Answer: B

Question 4: What would the values of `x` and `y` be after this segment of code?

```
def hi():
    print('Hello')

def greet():
    return 'Hello'

x = hi()
y = greet()
```

A. `x` would have the value `'Hello'`

`y` would have the value `'Hello'`

B. `x` would have the value `None`

`y` would have the value `'Hello'`

C. `x` would have the value `'Hello'`

`y` would have the value `None`

D. This would result in an error, because `hi()` doesn't return anything.

Answer: B

Quiz 12

Question 1: How would you write this function as a lambda function?

```
def add_two(x, y):
    return x + y
```

A. `lambda x, y: x + y`
B. `lambda x, y: return x + y`
C. `lambda add_two(return x + y)`

Answer: A

Question 2: How would you write this function as a lambda function?

```
def to_uppercase(name):
    return name.upper()
```

A. lambda to_uppercase(name): name.upper()
B. lambda name: name.upper()
C. lambda: arg.upper()

Answer: B

Question 3: What is a first-class function?

A. A function you can pass to another function as an argument. All functions in Python (both named and lambda) are first-class functions.
B. A very important function.
C. A function that can accept functions as arguments.

Answer: A

Question 4:

What would the output of this segment of code be?

It's a tricky one! Notice that the lambda function is defined inside the last line.

```
def over_age(data, getter):
    return getter(data) >= 18

user = { 'username': 'rolf123', 'age': '35' }

print(over_age(user, lambda x: int(x['age'])))
```

A. rolf123
B. True
C. False
D. It would give an error.

Answer: B

Section 4

Quiz 13

Question 1: How many objects of type Car do we have at the end of this code segment?

```
class Car:
    def __init__(self, make, model):
        self.make = make
        self.model = model

fiesta = Car('Ford', 'Fiesta')
focus = Car('Ford', 'Focus')
```

A. 1
B. 2
C. 4

Answer: B

Question 2: What sort of things can dictionaries and objects store?

A. A dictionary can store data; an object can store data and methods (functions that the object can execute with access to its own data).
B. They both store the same things.
C. A dictionary can store data and methods (functions that it can execute); an object can only store data.

Answer: A

Question 3: Given this class:

```
class Student:
  def __init__(self, name, grades):
    self.name = name
    self.grades = grades
```

And this object:

```
anne = Student('Anne', [56, 87, 99])
rolf = Student('Rolf', [80, 90, 100])

# print anne's grades here... how?
```

How would we print out Anne's grades?

A. print(grades)
B. print(self.grades)
C. print(anne.grades)
D. print(anne.self.grades)

Answer: C

Question 4: Would this be valid in Python?

```
class Movie:
  def __init__(current_object, name, year):
    current_object.name = name
    current_object.year = year
```

A. No, it doesn't have the self-parameter.
B. Yes, the first parameter is still the object, but you can call it whatever you want.

Answer: B

Quiz 14

Question 1: What's special about the `__init__` method?

A. Nothing.
B. It gets called immediately after the object is created.
C. It gets called before the object gets created.

Answer: B

Question 2: What's special about the `__len__` method?

A. We never call it manually as `my_object.__len__()`. Instead, we use the built-in function `len()`.

   When we call `len(my_object)` , that calls `my_object. __len__()`.

B. Nothing is special about this method, it just returns a number.
C. It can only be implemented in things that have a length or size.

Answer: A

Question 3: Given this code, what would the output of the segment be?

```
class Student:
    def __init__(self, name, school):
        self.name = name
        self.school = school
        self.marks = []

    @property
    def average(self):
        return sum(self.marks) / len(self.marks)

class WorkingStudent(Student):
    def __init__(self, name, school, salary):
        super().__init__(name, school)
        self.salary = salary


rolf = WorkingStudent("Rolf", "MIT", 15.50)
rolf.marks.append(57)

print(rolf.average)
```

A. An error, we forgot to add `()` after calling the `average` method.
B. 57
C. 57.0
D. An error, we only have 1 mark.

E. An error, WorkingStudent does not have an average() method.

Answer: C

Question 4: What methods must you define in order to be able to use a for loop in your class?

A. `__init__` and `__len__`
B. Just `__getitem__`
C. `__len__` and `__iter__`
D. You just need one method: `__for__`

Answer: B

Question 5: What is the difference between the `__repr__` and `__str__` magic methods?

A. `__repr__` should be used to return a string representing the object such that with that string you can re-create the object fully.

`__str__` should be used to return a string that is more suitable for users to read (e.g. it can be more descriptive but not have every little detail of the object).

B. `__repr__` should be used to return a string that is more suitable for users to read (e.g. it can be more descriptive but not have every little detail of the object).

`__str__` should be used to return a string representing the object such that with that string you can re-create the object fully.

C. They are the same.

Answer: A

Quiz 15

Question 1: All methods in a class must have `self` as the first parameter.

A. True.
B. False. All methods must have the object as the first parameter, but the parameter can be called anything we want.
C. False. You can have methods that don't take the object as the first parameter—such as `@classmethods` or `@staticmethods`.

Answer: C

Question 2: There are some cases where we don't need to use `self` in a method. For example, the `greet_friend()` method below:

```
1  class Person:
2      def __init__(self, name):
3          self.name = name
4
5      def greet_friend(self, friend_name):
6          return f'Hey there, {friend_name}!'
```

Which would be a better alternative, `@classmethod` or `@staticmethod`? Before answering, think carefully about why.

A. `@classmethod`
B. `@staticmethod`

Answer: B

Section 5

Quiz 16

Question 1: What is a traceback?

A. The entire exception: message and exception name.
B. The part of the error message that tells us the lines of code where the error happened, and the functions calls that happened.

Answer: B

Question 2: Once you've read through the traceback, what is the first thing we must do in order to start fixing an error?

A. Read through the code.
B. Use the debugger.
C. Search on the internet.

Answer; A

Question 3: Why can it be helpful to be familiar with some of the built-in exceptions in Python?

A. Built-in exceptions have descriptive names which can help us identify the source of the error quickly
B. It is not very useful; the traceback gives you more information.
C. Built-in exceptions never happen in larger programs, so it's not very useful to be familiar with them.

Answer: A

Quiz 17

Question 1: What is the correct syntax for creating a new error in Python?

A. except MyErrorClass
B. new Error()
C. raise MyErrorClass()

Answer: C

Question 2: Why would you ever create your own error classes in Python?

A. You wouldn't, you might as well use the built-in error classes.
B. It can be useful to add more descriptive names (e.g., `UserNotLoggedInError` , or `CarMalformedError`), so that finding the problem later on is easier.
C. It can be useful because we can easily change how exceptions are caught and dealt with.

Answer: B

Question 3: Errors can be used to make it explicit when something didn't work as intended, so that the caller can deal with the failure (asking for forgiveness).

Another option, less popular in Python, is to check whether the thing you're trying to do will succeed before doing it (asking for permission).

A. True
B. False

Answer: A

Quiz 18

Question 1: A quick question to warm you up!

Ben and Dan are two novice software developers. They are trying to build a function that asks user for a number and calculate its power of 2. Here is their first version of code:

```
def power_of_two():
    n = input('Please enter a number: ')
    n_square = n ** 2
    return n_square
```

What do you think of the code, will it work? Before submitting your answer, think about why.

A. Yes, it will work.
B. Yes, but it only works if the user inputs a number.
C. No, the program will break in all cases.

Answer: C

Question 2: Ben and Dan soon realized the error and here is their second version of code:

```
def power_of_two():
    user_input = input('Please enter a number: ')
    n = float(user_input)
    n_square = n ** 2
    return n_square
```

What do you think of the code, will it work now?

A. Yes, it will work since it now converts the input to a `float`.
B. Yes and no, the program will work on some input but break on others.
C. No, the program will still break anyway.

Answer: B

Question 3: After shipping their version 2 code, Ben and Dan had some serious complaints from the users. Here is Ben's quick fix:

```
def power_of_two():
    user_input = input('Please enter a number: ')
    try:
        n = float(user_input)
    except ValueError:
        print('Your input was invalid.')
    finally:
        n_square = n ** 2
        return n_square
```

Dan wants to test this function with some different inputs when the program asks for a number. He decides to test with two cases. First case: he enters 4, which is a valid numeric input. Second case: he enters 'dan' which is an invalid input.

What do you expect the function to return with Dan's input (Denoted as `[4, 'dan']`)? If the program breaks due to an error, we mark the function's return value as `Error`, and if it does not return anything, we mark the return value as `None`.

This one's a bit of a tricky one! Type the code out and try it in your editor for optimal chances of getting it right!

A. `[16.0, None]`
B. `[Error, Error]`
C. `[16.0, 0.0]`
D. `[16.0, Error]`
E. `None of the above.`

Answer: D

Question 4: As Dan found the error in the code, he decided to make the following change:

```
def power_of_two():
    user_input = input('Please enter a number: ')
    try:
        n = float(user_input)
    except ValueError:
        print('Your input was invalid. Using default value 0')
        n = 0
    else:
        n_square = n ** 2
        return n_square
```

Dan wants to test with the same test cases from last time: `[4, 'dan']`.

What do you expect the function return with Dan's input (denoted as `[4, 'dan']`) this time? If the program breaks due to an error, we mark the function's return value as Error, and if it didn't return anything, we mark the return value as `None`.

A. `[16.0, None]`
B. `[Error, Error]`
C. `[16.0, 0.0]`
D. `[16.0, Error]`
E. `None of the above.`

Answer: A

Question 5: Now that Dan's code still cannot work, Ben insisted on using a finally block, here is his code:

```
def power_of_two():
    user_input = input('Please enter a number: ')
    try:
        n = float(user_input)
    except ValueError:
        print('Your input was invalid. Using default value 0')
        n = 0
    else:
        n_square = n ** 2
    finally:
        return n_square
```

They decided to use the same test cases: `[4, 'dan']`.

What do you expect the function to return with the two input (denoted as `[4, 'dan']`) this time? If the program breaks due to an error, we mark the function's return value as `Error`, and if it does not return anything, we mark the return value as `None`.

A. `[16.0, None]`
B. `[Error, Error]`
C. `[16.0, 0.0]`
D. `[16.0, Error]`
E. None of the above.

Answer: D

Question 6: Ben and Dan just won't give up, here's their last try:

```python
def power_of_two():
    user_input = input('Please enter a number: ')
    try:
        n = float(user_input)
    except ValueError:
        print('Your input was invalid. Using default value 0')
        return 0
    finally:
        n_square = n ** 2
        return n_square
```

They decided to use the same test cases: `[4, 'dan']`.

What do you expect the function to return with the two input (denoted as `[4, 'dan']`) this time? If the program breaks due to an error, we mark the function's return value as `Error`, and if it does not return anything, we mark the return value as `None`.

A. `[16.0, None]`
B. `[16.0, 0.0]`
C. `[16.0, 0]`
D. `[16.0, Error]`
E. None of the above

Answer: D

Quiz 19

Question 1: What is the purpose of re-raising an error once you've caught it?

A. There is no point.
B. To do something now an error has happened, but also notify other callers that something critical may have happened.

   For example, we could decide to log the error but then continue bubbling it up the function calls.

Answer: B

Question 2: try-except blocks have a lesser known brother; the else block.

This block is executed after an exception happens and it has been handled.

A. True
B. False

Answer: B

Question 3: try-except blocks have another brother block, the finally block.

This block gets executed after the exception has been handled—but also if an exception doesn't occur.

A. True
B. False

Answer: A

Question 4: What is the purpose of debugging?

A. Real programmers don't debug.
B. To step through the code and easily notice values of variables and results of functions.
C. To find bugs.

Answer: B

Section 9

Quiz 20

Question 1: All generators in Python are Iterators.

A. True
B. False

Answer: A

Question 2: All iterators in Python are generators.

A. True
B. False

Answer: B

Question 3: All iterators are iterables.

A. True
B. False

Answer: B

Question 4: As we know, the `filter()` function takes in two arguments, the first one is a "decision function" and the second argument is the object that we want to filter on. So how does filter() decide whether to choose an item?

A. The `filter()` function chooses an item only if the "decision function" returns `True`.
B. The `filter()` function chooses an item only if the "decision function" returns any value that evaluates to True (i.e. it doesn't have to be the `True` keyword specifically).
C. The `filter()` function chooses an item only if the "decision function" returns `False`.
D. The `filter()` function chooses an item only if the "decision function" returns any value that evaluates to `False`.

Answer: B

Question 5: Consider the following variable `b`, would it evaluate to `True` or `False` in an if statement?

```
b = -0.0
```

A. True
B. False

Answer: B

Question 6: Consider the following variable `b`, does it evaluate to `True` or `False`?

```
b = 0.000000000000000000000001
```

A. True
B. False

Answer: A

Question 7: Consider the following variable `b`, does it evaluate to `True` or `False`?

```
b = []
```

A. True
B. False

Answer: B

Question 8: Consider the following variable `b`, does it evaluate to `True` or `False`?

```
b = [None]
```

A. True
B. False

Answer: A