`Directions:` The exam is 120 minutes long. Please read each question carefully.

When asked to write code, you should write working Python code that has correct syntax. You should explain in 1-2 sentences what the idea for your solution is or write next to your code what it is doing. This will increase your chances of getting full/partial credit.
Use the backs of the pages if needed.

Last Name: _____

First Name: _____

Student ID #: _____

Question	Points	Score
1	20	
2	20	
3	20	
4	20	
5	20	
6	20	
7	20	
8	20	
Total:	160	

1. (20 points) Write down the output of the following programs.

   1.
```
x = 1
s = 0
for i in range(8):
    s += x
    x += 1
print(s)
```

   Answer: This computes $1 + 2 + \cdots + 7 = 36$

   2.
```
def f(n):
    if n > 0:
        return n * g(n)
    return 1

def g(n):
    return f(n // 2)

print(f(6))
```

   Answer: $f(6) = 6 * g(6)$, $g(6) = f(3)$, $f(3) = 3 * g(3)$, $g(3) = f(1)$, $f(1) = 1 * g(1)$, $g(1) = f(0)$, $f(0) = 1$. So $f(6) = 6 * 3 * 1 = 18$

   3.
```
from functools import reduce
x = reduce(lambda a,d: 2*a+d, [1,0,0,0,0,1,0,1])
print(x)
```

   Answer: Recall how the reduce function works. You apply the function to the first two elements, then apply the function to the answer and the next element,... In this case you keep multiplying by two and adding. The answer is 133. If you think about it, you will notice that this computes the base 10 version of the binary number.

   4.
```
def f(xs):
    if xs == []:
        return 0
    return xs[0] + f(xs[1:])

f([1,2,3,4,5])
```

   Answer: `f([1,2,3,4,5] = 1 + f([2,3,4,5]))`,
   `f([2,3,4,5]) = 2 + f([3,4,5])`,... It's computing the sum. Which is 15.

2. (20 points) Produce the following lists without using for or while loops.

   1. `[0, 1, 3, 7, 15, 31, 63, 127, 255, 511]`

      Solution: Powers of two minus 1.

      ```
      [2 ** x - 1 for x in range(10)]
      ```

   2. `[1, 2, 4, 5, 7, 8, 10, 11, 13, 14, 16, 17, 19]`

      Solution: all the numbers except multiples of 3.

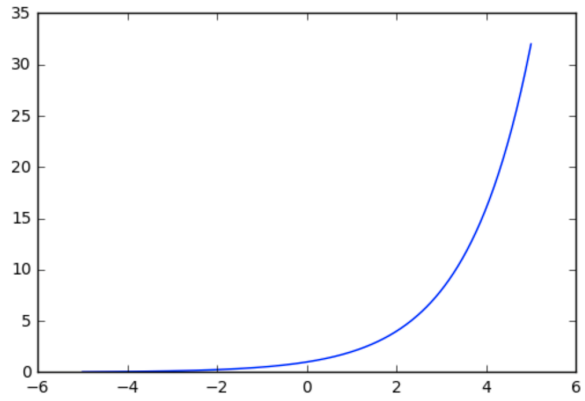
      ```
      [x for x in range(20) if x % 3 != 0]
      ```

   3. `[-1, 2, -3, 4, -5, 6, -7, 8, -9, 10, -11, 12, -13, 14]`

      Solution: all the numbers in range but multiply by alternating sign.

      ```
      [x * (-1)**x for x in range(1,15)]
      ```

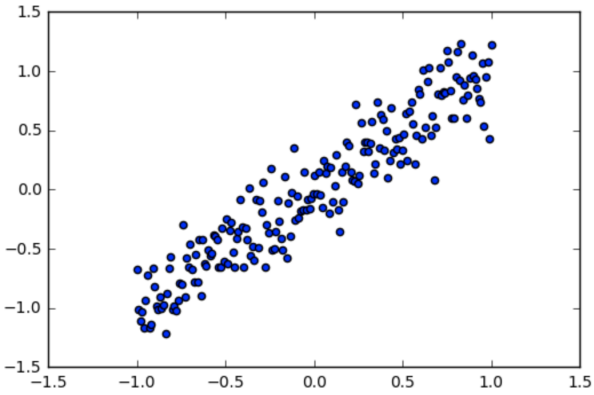
3. (20 points) Write code that will produce the following graphs (or something that looks like it; use `plt.plot(X, Y)` and `plt.scatter(X, Y)`).

1. 

Solution: exponential function.

```
X = np.linspace(-5,5,400)
Y = 2 ** X
plt.plot(X, Y)
```

(`a**X` for any other `a` would receive full credit too)

2. 

Solution: adding some noise to $y = x$.

```
X = np.linspace(-1,1,200)
Y = X + np.random.normal(0,0.2,X.shape[0])
plt.scatter(X, Y)
```

(Of course, the 0.2 doesn't matter as long as you knew how to add some noise.)

4. (20 points) Complete the code below to implement the function `chessboard(n)` that will return a numpy array with 1's and 0's arranged in a chessboard pattern. Assume `n` is odd. Examples:

```
In:   chessboard(3)
Out:  array([[0, 1, 0],
             [1, 0, 1],
             [0, 1, 0]])

In:   chessboard(5)
Out   array([[0, 1, 0, 1, 0],
             [1, 0, 1, 0, 1],
             [0, 1, 0, 1, 0],
             [1, 0, 1, 0, 1],
             [0, 1, 0, 1, 0]])
```

Solution: reshape an array with alternating 0's and 1's.

```
def chessboard(n):
    X = np.array([x % 2 for x in range(n * n)]).reshape([n,n])
    return X
```

Complete the code below to implement the function `chessgonewrong(n)`, which produces a chess-board with the middle 3×3 square having -1's instead of 1s.

```
In:   chessgonewrong(7)
Out:  array([[ 0,  1,  0,  1,  0,  1,  0],
             [ 1,  0,  1,  0,  1,  0,  1],
             [ 0,  1,  0, -1,  0,  1,  0],
             [ 1,  0, -1,  0, -1,  0,  1],
             [ 0,  1,  0, -1,  0,  1,  0],
             [ 1,  0,  1,  0,  1,  0,  1],
             [ 0,  1,  0,  1,  0,  1,  0]])
```

```
def chessgonewrong(n):
    X = chessboard(n)

    return X
```

Solution: multiply a 3×3 square at the center by -1.

```
def chessgonewrong(n):
    X = chessboard(n)
    X[n // 2 - 1 : n // 2 + 2, n // 2 - 1 : n // 2 + 2] *= -1
    return X
```

5. (20 points) Implement a function `divisors(n)` that returns all positive integer divisors of an integer `n` as a list. (returns not prints)

Solution: Loop through i=1,...,n and add them to the list if they divide.

```
def divisors(n):
    xs = []
    for i in range(1,n+1):
        if n % i == 0:
            xs.append(i)
    return xs
```

6. (20 points) A palindrome is a word that is the same when reversed, e.g. "amanaplanacanalpanama". Write a function `ispalin(s)` that will return `True` if a string `s` is a palindrome and `False` otherwise. (remark: you can work with `s` as if it were a list).

Solution: as you loop to the middle, check to see if the character matches with the corresponding character from the other side. If there are any mismatches, return False. If you finish the loop without any mismatches, then return True.

```
def ispalin(s):
    n = len(s)
    for i in range(n//2):
        if s[i] != s[n-1-i]:
            return False
    return True
```

7. (20 points) Recall the `Polynomial` class from the homework that stores a polynomial as a list of its coefficients. Implement the `__add__(self, other)` function that returns a new polynomial which represents the sum of the polynomials `self` and `other`.

```
class Polynomial():
    def __init__(self, xs):
        self.coeffs = xs

    # returns a string representation of the polynomial
    def __repr__(self):
        if self.coeffs == []:
            return "0"
        c = ""
        for i, x in enumerate(self.coeffs):
            c += str(x) + "x" + "^" + str(i) + " + "
        return c[:-3]

    def __add__(self, other):
```

Solution: This was a problem in Homework 5. You add the corresponding coefficients (but have to be careful about not getting index out of bounds errors because you are looking too far to the right).

```
    def __add__(self, other):
        i = 0
        new_coeffs = [0.0 for i in range(max(len(self.coeffs), len(other.
            coeffs)))]
        for i in range(len(new_coeffs)):
            coefa = self.coeffs[i] if i < len(self.coeffs) else 0.0
            coefb = other.coeffs[i] if i < len(other.coeffs) else 0.0
            new_coeffs[i] = coefa + coefb
        return Polynomial(new_coeffs)
```

8. (20 points) Write code that will find the minimum of the function $f(x, y) = x^4 + y^2 + 2x + 4y + 1$ using gradient descent. (Start the descent from $(x, y) = (5, 5)$ and use the learning rate of $\eta = 0.01$). Your code should print the minimum value it finds.

Solution: Please refer to the lecture notes on gradient descent for a careful explanation.

```
f = lambda x, y: x**4 + y**2 + 2*x + 4*y + 1
dfdx = lambda x, y: 4 * x**3 + 2
dfdy = lambda x, y: 2*y + 4

num_steps = 1000
eta = 0.01
x = 5
y = 5
for i in range(num_steps):
    # compute change in x and y BEFORE changing them
    dx = eta * dfdx(x,y)
    dy = eta * dfdy(x,y)
    x -= dx
    y -= dy

print(x,y,f(x,y))
```