Python for Math and Stat Fall 2023
Final Exam

Assume that all necessary packages have been imported.

1. (15 pts) For the following 4 problems, write down what each code block would display if executed in a Jupyter cell. If the code generates an error or infinite loop, write `Error`. Assume

```
arr = np.array([[10,  1,  9,  7, 12],
                [ 3, 16,  5,  8,  9]])
```

   (a)
```
arr[1, 4:0:-2]
```

   (b)
```
arr[arr > 9]
```

   (c)
```
kvals = {k: k*k for k in range(10)}
kvals[kvals[2]]
```

   (d)
```
def func(nums):
    print(nums, end=' ')
    if len(nums) == 1:
        return 10
    else:
        return func(nums[1:]) + 4

vals = [-5, 6, 2]
func(vals)
```

Solution:

(a)
```
array([ 9, 5])
```

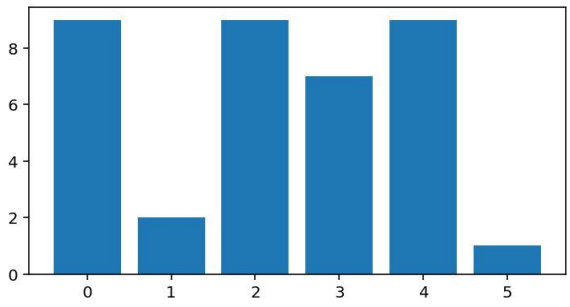
(b)
```
array([10, 12, 16])
```

(c)
```
16
```

(d)
```
[-5, 6, 2] [6, 2] [2]
18
```

2. (8 pts) Write a function **`plot_digits(num)`** that plots the digits of a positive integer as a bar chart. It does not return a value. Use **`try except`** to catch invalid input, in which case no plot is created; `Error` is displayed instead.

   Examples:
   `plot_digits('abc')` displays `Error`.
   `plot_digits(929791)` produces the following result:



Solution:

```
def plot_digits(num):
    try:
        digits = [int(d) for d in str(num)]
        plt.bar(range(len(digits)), digits)
        plt.show()

    except:
        print('ERROR')
```

3. (10 pts) To check for errors when scanning or manually entering product bar codes, an extra check digit is included.

Here is a procedure for calculating the check digit for an integer code: (ex: 15694)

- Add every other digit in the code, starting with the first digit. (ex: $\underline{1}\,5\,\underline{6}\,9\,\underline{4}$: $1 + 6 + 4 = 11$)
- Add every other digit in the code, starting with the second digit. (ex: $1\,\underline{5}\,6\,\underline{9}\,4$: $5 + 9 = 14$)
- Add the second sum to 3 times the first sum. (ex: $14 + 3 \cdot 11 = 47$)
- The units digit of the result is the check digit. (ex: 7)

Write a function **`check_digit(code)`** that takes an integer `code` greater than 9 and returns its check digit as an `int`. For example, `check_digit(15694)` returns 7.

Solution:

```
def check_digit(code):
    strcode = str(code)

    sum0 = sum(int(d) for d in strcode[0::2])
    sum1 = sum(int(d) for d in strcode[1::2])

    total = sum1 + 3*sum0

    return total % 10
```

4. (10 pts) Consider the polynomial

$$P(x) = 1 + 2x + 3x^2 + 4x^3 + \cdots + nx^{n-1}.$$

Write a function **`poly_eval(x, n)`** that calculates the value of $P(x)$ given values for x and positive integer n. Use **numpy** features (such as `arange` and vectorization). *Do not* include a loop.

Example: `poly_eval(2, 3)` returns `17` which equals $1 + 2(2) + 3(2)^2$.

Solution:

```
def poly_eval(x, n):
    exponents = np.arange(n)
    coeffs = exponents + 1
    return np.sum(coeffs * x**exponents)
```

5. (12 pts) The DataFrame **`dfcocoa`**, shown below, contains information about various cocoa powder products. Each row provides the name, weight (in ounces), and price (in dollars) for a distinct product.

Product	Ounces	Price
Droste	8.8	9.65
Anthonys	32.0	17.99
...		
Valrhona	8.8	15.49
Ghirardelli	48.0	20.75

Write code to do the following:

(a) Add a new `Nestle` product to the DataFrame with a weight of 8 ounces and a price of 2.75 dollars.

(b) Add a new column to the DataFrame called `UnitPrice` which equals the price per ounce for each product.

(c) Select the names of all products with a `UnitPrice` greater than the unit price for `Hersheys` (which is a product in `dfcocoa`). The result should be a pandas index or a list of strings.

(d) One of the products has the lowest unit price. Identify the name of that product as a string.

Solution:

(a) `dfcocoa.loc['Nestle'] = [8, 2.78]`

(b) `dfcocoa['UnitPrice'] = dfcocoa.Price / dfcocoa.Ounces`

(c) `dfcocoa[dfcocoa.UnitPrice > dfcocoa.UnitPrice['Hersheys']].index`

(d) `dfcocoa.UnitPrice.idxmin()`

6. (20 pts) Create a class called **`Coin`**. Each instance of the class represents a coin with one attribute:

   - **`prob_H`**: probability of flipping a head. Assume that `prob_H` is a value between 0 and 1. Set the **default** value to 0.5.

   and these methods:

   - **`flip()`**: returns `'H'` or `'T'` given probability `prob_H`. (For example, if `prob_H` equals 0.2, then out of 100 flips, `'H'` will appear about 20 times.)
   - **`flip_until(outcome)`**: simulates the flipping of the coin, printing the results in a row, until the desired outcome appears. Return the number of flips. Assume that `outcome` is either `'H'` or `'T'`. This method should call `flip()`.

     Example: `flip_until('T')` might print `HHHHT` and return `5`.

   Solution:

```
class Coin:
    def __init__(self, prob_H=0.5):
        self.prob_H = prob_H

    def flip(self):
        if random.random() < self.prob_H:
            return 'H'
        else:
            return 'T'

    def flip_until(self, outcome):
        nflip = 0
        result = ''

        while result != outcome:
            result = self.flip()
            print(result, end='')
            nflip += 1

        return nflip
```