

## Tarea 2

### Mario Moreno Zamora

Para esta tarea se siguen casi mismos lineamientos empleados en la Tarea de Vacaciones, es decir, se debe crear un directorio que se llama "Tarea Semana 2" y dentro debe almacenarse este notebook con la nomenclatura con la salvedad de que si se trata de una pregunta de implementación se debe colocar en ese mismo ejercicio la resolución y si se aborda una pregunta abierta se debe colocar la fuente.

De estas últimas todas deben colocarse en Markdown (siguiendo los lineamientos de visualización) o de lo contrario no se valdrá el crédito.¶

```
#Ejercicio 1
#A partir de la siguiente matriz:

matriz = [[1,2,3,4,5],
          [6,7,8,9,10],
          [11,12,13,14,15],
          [16,17,18,19,20]]

"""
Imprima los elementos de esta manera:

1 2 3 4 5
6 7 8 9 10
11 12 13 14 15
16 17 18 19 20

Usando solamente un ciclo (while, for); no hay restricción de if-else,
breaks, etc.
```

```
Result_t = [k for k in range (1,21)]
```

```
Print (result_t)
```

```
Result_t = []
```

### #Ejercicio 3

"""

*En general en los lenguajes de programación una de las prácticas recomendadas para nombrar ya sea variables o funciones es el "Camel Case" el cual consiste en sustituir posibles espacios con letras mayúsculas (la primera letra de la frase por lo regular no se toma en cuenta), por ejemplo:*

*este es el nombre de mi variable => esteEsElNombreDeMiVariable  
aquí está otro nombre para la variable => aquíEstaOtroNombreParaLaVariable*

*¿Por qué Camel? porque la combinación de mayúsculas y minúsculas simula la joroba de un camello (dato curioso debería de ser un dromedario y no un camello porque el camello tiene una joroba y el dromedario dos ;))*

*Entonces, para este ejercicio teniendo en cuenta esta cadena:*

*En algún lugar de La Mancha de cuyo nombre no quiero acordarme, o ha mucho tiempo que vivía un hidalgo de los de lanza en astillero, adarga antigua, rocín flaco y galgo corredor.*

*Se debe convertir bajo la nomenclatura Camel Case. Para este ejercicio en particular la primera palabra de la frase se deja intacta mientras que los signos de puntuación (exceptuando el último) se deben remover.*

*Sugerencias: las funciones lower() & upper() convierten una letra a minúsculas y mayúsculas respectivamente, si se da el caso de que ya estaban convertidas a los tipos que se deseaba transformar no arroja error.*

*Por otra parte las funciones `islower()` & `isupper()` indican si una letra está en su modalidad minúscula o mayúscula respectivamente.*

```
cadena = "En algun lugar de La Mancha de cuyo nombre no quiero acordarme, o ha mucho tiempo que vivia un hidalgo de los de lanza en astillero, adarga antigua, rocin flaco y galgo corredor."
```

**Cadena = "En algun lugar de La Mancha de cuyo nombre no quiero acordarme, o ha mucho tiempo que vivia un hidalgo de los de lanza en astillero, adarga antigua, rocin flaco y galgo corredor."**

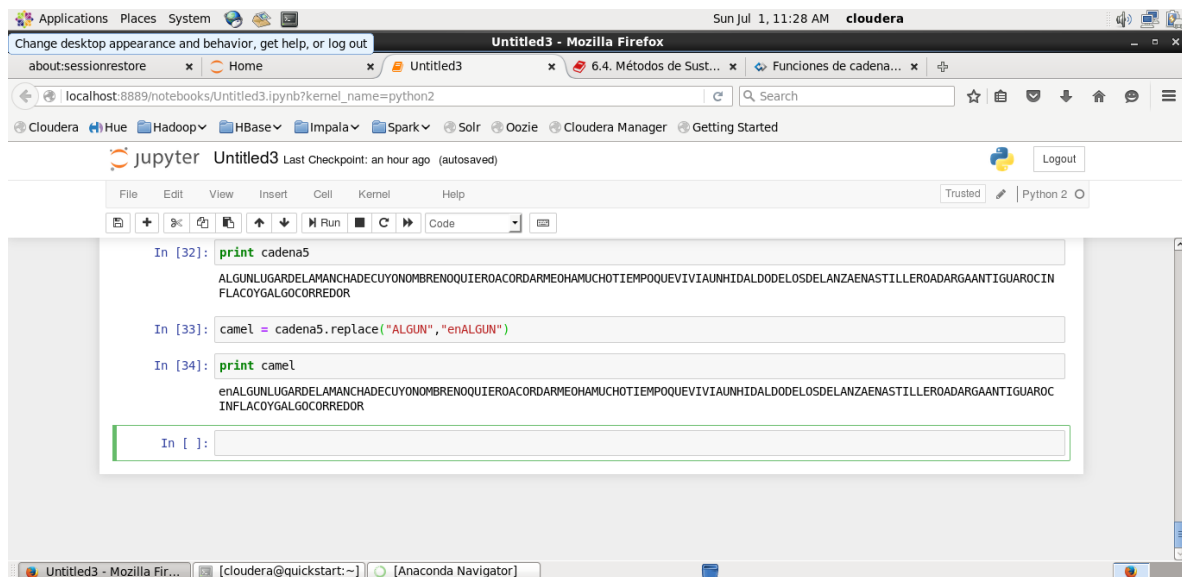
**Cadena2 = cadena.upper()**

**Cadena3=cadena2.replace(","," ")**

**Cadena4 = cadena3.replace(" ","")**

**Cammel = cadena4.replace("ENALGUN","enALGUN")**

**Print cammel**



#### #Ejercicio 4

"""

Se define "raíz digital" a la suma recursiva de todos los dígitos de un número hasta llegar a uno solo, por ejemplo

`raiz_digital(16)`

`=> 1 + 6`

`=> 7`

`raiz_digital(942)`

`=> 9 + 4 + 2`

`=> 15 ...`

`=> 1 + 5`

`=> 6`

Entonces el objetivo consiste en implementar el algoritmo de la raíz digital y probar con el número 493193, cuyo resultado debe ser 2. Sugerencia: la forma recursiva es la más fácil de implementar.

Curso rápido para ingresar funciones en Python; basta con declarar:

#En def no hay tipo de retorno de dato como en otros lenguajes.

`def nombre_funcion(parametro_1,...,parametro_n):`

`#El return del tipo de dato no cuenta en Python.`

`return algo`

"""

**Def raíz\_digital (numero):**

**Suma = sum (int(digito) for digito in str (numero))**

**If suma < 10:**

**Return suma**

**Else:**

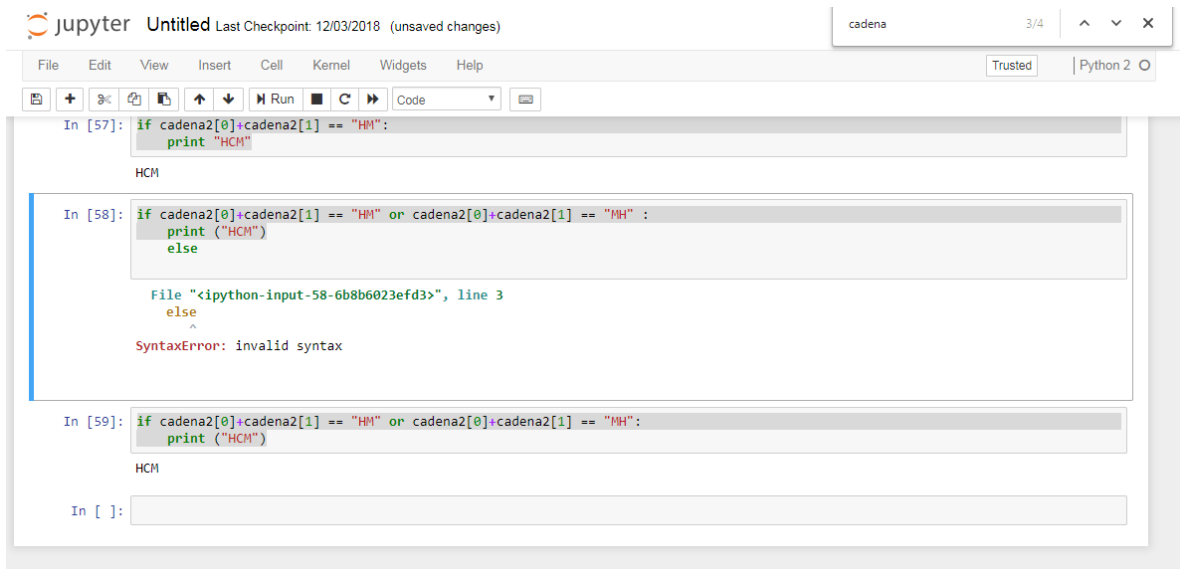
**Return raíz\_digital (suma)**

**Print raíz\_digital (493193)**



```
"""
```

```
if cadena2[0]+cadena2[1] == "HM" or cadena2[0]+cadena2[1] == "MH":  
    print ("HCM")
```



```
Jupyter Untitled Last Checkpoint: 12/03/2018 (unsaved changes)  
cadena 3/4  
File Edit View Insert Cell Kernel Widgets Help  
In [57]: if cadena2[0]+cadena2[1] == "HM":  
         print ("HCM")  
HCM  
In [58]: if cadena2[0]+cadena2[1] == "HM" or cadena2[0]+cadena2[1] == "MH":  
         print ("HCM")  
         else  
         File "<ipython-input-58-6b8b6023efd3>", line 3  
           else  
           ^  
SyntaxError: invalid syntax  
In [59]: if cadena2[0]+cadena2[1] == "HM" or cadena2[0]+cadena2[1] == "MH":  
         print ("HCM")  
HCM  
In [ ]:
```

#Ejercicio 9

```
"""
```

*Con base en el archivo ejercicio\_5.txt de la tarea anterior cree un dataframe de Pandas considerando el encabezado, por ahora no se preocupe del tipo de dato.*

```
"""
```

```
import pandas as pd  
  
dataframe = pd.read_csv('C:\Users\Mario Moreno Zamora\Downloads\ejercicio_5.txt')  
  
dataframe.describe()
```

jupyter Untitled Last Checkpoint: 12/03/2018 (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 2

In [6]: `dataframe = pd.read_csv('C:\Users\Mario Moreno Zamora\Downloads\ejercicio_5.txt')`

In [8]: `dataframe.describe()`

Out[8]:

	Year	Month	DayOfMonth	DayOfWeek	DepTime	CRSDepTime	ArrTime	CRSArrTime	FlightNum	ActualElapsedTime	...	TaxiIn
count	35.000000	35.000000	35.000000	35.000000	35.000000	35.000000	35.000000	35.000000	35.000000	35.000000	...	26.000000
mean	2002.600000	11.485714	14.600000	5.571429	954.314286	944.028571	1163.000000	1135.657143	1581.514286	106.971429	...	10.653846
std	9.312231	0.886879	3.246718	1.170362	347.753866	346.644036	371.983792	387.247674	78.286191	36.205692	...	6.305187
min	1987.000000	10.000000	13.000000	1.000000	547.000000	545.000000	646.000000	650.000000	1451.000000	43.000000	...	4.000000
25%	1997.500000	11.000000	13.000000	6.000000	729.000000	730.000000	902.500000	849.000000	1531.500000	84.000000	...	6.250000
50%	2008.000000	12.000000	13.000000	6.000000	810.000000	815.000000	1024.000000	1008.000000	1621.000000	97.000000	...	8.000000
75%	2008.000000	12.000000	13.500000	6.000000	1071.500000	1072.000000	1429.500000	1427.500000	1631.500000	122.500000	...	13.750000
max	2008.000000	12.000000	24.000000	7.000000	1910.000000	1910.000000	2017.000000	2016.000000	1641.000000	234.000000	...	24.000000

8 rows x 25 columns

In [ ]:

In [ ]:

#Ejercicio 10

"""Tome el dataframe de Pandas del ejercicio anterior, guárdelo como un archivo de tipo .pickle con el nombre: <su\_usuario\_de\_git> y colóquelo en el subdirectorio "Tarea Semana 2" """

import pickle

`pickle.dump(dataframe, open("dataframe.p", "wb"))`

25%	1997.500000	11.000000	13.000000	6.000000	729.000000	730.000000	902.500000	849.000000	1531.5000
50%	2008.000000	12.000000	13.000000	6.000000	810.000000	815.000000	1024.000000	1008.000000	1621.0000
75%	2008.000000	12.000000	13.500000	6.000000	1071.500000	1072.000000	1429.500000	1427.500000	1631.5000
max	2008.000000	12.000000	24.000000	7.000000	1910.000000	1910.000000	2017.000000	2016.000000	1641.0000

8 rows x 25 columns

In [9]: `import pickle`

In [10]: `pickle.dump(dataframe, open("dataframe.p", "wb"))`

In [ ]:

#Ejercicio 11

"""

Para este ejercicio se debe implementar un programa que tome el lenguaje natural y lo convierta a código Morse:

[https://www.electronics-notes.com/articles/ham\\_radio/morse\\_code/characters-table-chart.php](https://www.electronics-notes.com/articles/ham_radio/morse_code/characters-table-chart.php)

Usando solamente un ciclo for/while; if-else y demás no tienen limitantes.

Para las pruebas de traducción exitosa se puede ocupar este ejemplo:

Creo que aquello en lo que nos convertimos depende de lo que nuestros padres nos enseñan en pequeños momentos, cuando no están intentando enseñarnos. Estamos hechos de pequeños fragmentos de sabiduría.

Y el siguiente sitio:

[https://morsecode.scphillips.com/translator.html?utm\\_source=hootsuite](https://morsecode.scphillips.com/translator.html?utm_source=hootsuite)

NOTA: puede deshacerse de los signos de puntuación pero no de las letras acentuadas, por otro lado las mayúsculas y minúsculas se comportan igual (upper(), lower()). En este caso puede convertir las letras "á" ó "à" o "â" por "a" y la "ñ" por "n".

```
# Python program to implement Morse Code Translator

'''
VARIABLE KEY
'cipher' -> 'stores the morse translated form of the english string'
'decipher' -> 'stores the english translated form of the morse string'
'citext' -> 'stores morse code of a single character'
'i' -> 'keeps count of the spaces between morse characters'
'message' -> 'stores the string to be encoded or decoded'
'''

# Dictionary representing the morse code chart
MORSE_CODE_DICT = { 'A':'.-.', 'B':'-...',
                    'C':'-.-.', 'D':'-..', 'E':'.',
                    'F':'.-.-.', 'G':'--.', 'H':'.....',
                    'I':'...', 'J':'.---', 'K':'-.-',
                    'L':'.-..', 'M':'--', 'N':'-.',
                    'O':'---', 'P':'.--.', 'Q':'--.-',
                    'R':'.-.', 'S':'....', 'T':'-',
                    'U':'.-.', 'V':'....-', 'W':'.-.-',
                    'X':'-.-.', 'Y':'-.-.-', 'Z':'--...',
                    '1':'.-.-.-', '2':'.-.-.-', '3':'.-.-.-',
                    '4':'.-.-.-', '5':'.-.-.-', '6':'.-.-.-',
                    '7':'.-.-.-', '8':'.-.-.-', '9':'.-.-.-',
                    '0':'.-.-.-', ' ':'.-.-.-', '.':'.-.-.-',
                    '?':'.-.-.-', '/':'.-.-.-', '-':'.-.-.-',
                    '(':'.-.-.-', ')':'.-.-.-' }

# Function to encrypt the string
```



```

# according to the morse code chart
def encrypt(message):
    cipher = ''
    for letter in message:
        if letter != ' ':

            # Looks up the dictionary and adds the
            # corresponding morse code
            # along with a space to separate
            # morse codes for different characters
            cipher += MORSE_CODE_DICT[letter] + ' '
        else:
            # 1 space indicates different characters
            # and 2 indicates different words
            cipher += ' '

    return cipher

# Function to decrypt the string
# from morse to english
def decrypt(message):

    # extra space added at the end to access the
    # last morse code
    message += ' '

    decipher = ''
    citext = ''
    for letter in message:

        # checks for space
        if (letter != ' '):

            # counter to keep track of space
            i = 0

            # storing morse code of a single character
            citext += letter

            # in case of space
            else:
                # if i = 1 that indicates a new character
                i += 1

            # if i = 2 that indicates a new word
            if i == 2:

                # adding space to separate words
                decipher += ' '
            else:

                # accessing the keys using their values (reverse of
                # encryption)
                decipher +=
list(MORSE_CODE_DICT.keys())[list(MORSE_CODE_DICT
                                .values()).index(citext)]
                citext = ''

```

```

        return decipher

# Hard-coded driver function to run the program

def main():

    message = "WE LOVE BIG DATA"

    result = encrypt(message.upper())

    print (result)

message = "-- .- .-. .. --- -- --- .-. . - . --- ---.. .- -- --- .-.
        .- "

    result = decrypt(message)

    print (result)

# Executes the main function

if __name__ == '__main__':

    main()

```

In [26]: # Python program to implement Morse Code Translator

```

...
VARIABLE KEY
'cipher' -> 'stores the morse translated form of the english string'
'decipher' -> 'stores the english translated form of the morse string'
'citext' -> 'stores morse code of a single character'
'i' -> 'keeps count of the spaces between morse characters'
'message' -> 'stores the string to be encoded or decoded'
...

# Dictionary representing the morse code chart
MORSE_CODE_DICT = { 'A': '-.-', 'B': '...-.',
                    'C': '-.-.-', 'D': '-.-.', 'E': '.-',
                    'F': '.-.-.-', 'G': '--.-', 'H': '....-',
                    'I': '.-.-', 'J': '-.-.-.-', 'K': '-.-.-',
                    'L': '.-..', 'M': '---', 'N': '---.',
                    'O': '---', 'P': '---.-', 'Q': '---.-.-',
                    'R': '---.', 'S': '...-', 'T': '-.-',
                    'U': '....', 'V': '....-', 'W': '---.-',
                    'X': '---.-.-', 'Y': '---.-.-', 'Z': '---.-.-',
                    '1': '-----', '2': '-----', '3': '-----',
                    '4': '-----', '5': '-----', '6': '-----',
                    '7': '-----', '8': '-----', '9': '-----',
                    '0': '-----', ' ': '-----', ' ': '-----',
                    '?': '-----', '/': '-----', '-': '-----',
                    '(': '-----', ')': '-----'}

```

```

# Function to encrypt the string
# according to the morse code chart
def encrypt(message):
    cipher = ''
    for letter in message:
        if letter != ' ':
            # Looks up the dictionary and adds the
            # corresponding morse code
            # along with a space to separate
            # morse codes for different characters
            cipher += MORSE_CODE_DICT[letter] + ' '
        else:
            # 1 space indicates different characters
            # and 2 indicates different words
            cipher += ' '

    return cipher

# Function to decrypt the string
# from morse to english
def decrypt(message):
    # extra space added at the end to access the
    # last morse code
    message += ' '

```

```

decipher = ''
citext = ''
for letter in message:
    # checks for space
    if (letter != ' '):
        # counter to keep track of space
        i = 0

        # storing morse code of a single character
        citext += letter

    # in case of space
    else:
        # if i = 1 that indicates a new character
        i += 1

        # if i = 2 that indicates a new word
        if i == 2 :
            # adding space to separate words
            decipher += ' '
        else:

```

```

        # accessing the keys using their values (reverse of encryption)
        decipher += list(MORSE_CODE_DICT.keys())[list(MORSE_CODE_DICT
        .values()).index(citext)]
        citext = ''

    return decipher

# Hard-coded driver function to run the program
def main():
    message = "WE LOVE BIG DATA"
    result = encrypt(message.upper())
    print (result)

    message = "-- .- .- .- --- -- .- .- --- -- .- .- --- --"
    result = decrypt(message)
    print (result)

# Executes the main function
if __name__ == '__main__':
    main()

```

```

..- .- .- .- --- -- .- .- --- -- .- .- --- --
MARIO MORENO ZAMORA

```

Fuente:

[http://localhost:8888/notebooks/Untitled.ipynb?kernel\\_name=python2](http://localhost:8888/notebooks/Untitled.ipynb?kernel_name=python2)

#Ejercicio 12

"""

*Para repasar el uso de funciones recursivas, implemente la función de Ackerman. Muestre en pantalla el resultado de :*

*ackerman(3,4)*

*NOTA: puede crear una función (precisamente llamada ackerman) para una mejor presentación del código.*

"""

```
def ackermann(m,n):
```

```
    if m == 0:
```

```
        return (n + 1)
```

```
    elif n == 0:
```

```
        return ackermann(m - 1, 1)
```

```
    else:
```

```
        return ackermann(m - 1, ackermann(m, n - 1))
```

```
x=int(input("What is the value for m? "))
```

```
    print x
```

```
y=int(input("What is the value for n? "))
```

```
    print y
```

```
print "\nThe result of your inputs according to the Ackermann Function is:"
```

```
    print (ackermann(3, 4))
```

```
def ackermann(m,n):
    if m == 0:
        return (n + 1)
    elif n == 0:
        return ackermann(m - 1, 1)
    else:
        return ackermann(m - 1, ackermann(m, n - 1))

x=int(input("What is the value for m? "))
print x

y=int(input("What is the value for n? "))
print y

print "\nThe result of your inputs according to the Ackermann Function is:"
print (ackermann(x, y))
```

```
What is the value for m? 3
3
What is the value for n? 4
4
```

```
The result of your inputs according to the Ackermann Function is:
125
```

Fuente: <http://pythonfiddle.com/ackermanns-function/>

#Ejercicio 13.

"""

*Investigue e implemente la criba de Eratóstenes; hágase valer de todas las funcionalidades vistas*

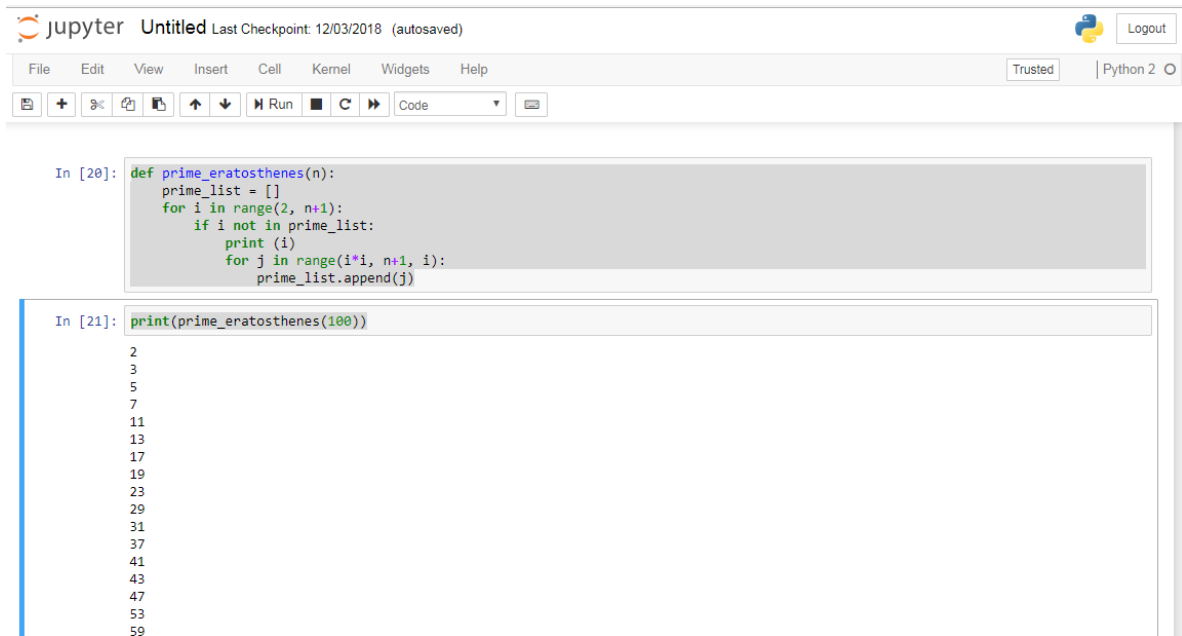
*hasta ahora (if-else, while, for, diccionarios, listas, etc.)*

*NOTA: para este ejercicio debe colocar 2 casillas, una para la explicación del tema EN MARKDOWN*

*y la otra para la implementación del mismo.*

"""

```
def prime_eratosthenes(n):
    prime_list = []
    for i in range(2, n+1):
        if i not in prime_list:
            print (i)
            for j in range(i*i, n+1, i):
                prime_list.append(j)
    print(prime_eratosthenes(100))
```



The image shows a Jupyter Notebook interface. At the top, the title bar says "jupyter Untitled" with a "Last Checkpoint: 12/03/2018 (autosaved)" and a "Logout" button. Below the title bar is a menu bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help". To the right of the menu bar are "Trusted" and "Python 2" buttons. Below the menu bar is a toolbar with icons for saving, adding cells, undo, redo, and running code. The main area contains two code cells. The first cell, labeled "In [20]:", contains a function definition for a prime sieve. The second cell, labeled "In [21]:", contains a call to the function with the argument 100. The output of the second cell is a list of prime numbers from 2 to 59.

```
In [20]: def prime_eratosthenes(n):
          prime_list = []
          for i in range(2, n+1):
              if i not in prime_list:
                  print (i)
                  for j in range(i*i, n+1, i):
                      prime_list.append(j)
```

```
In [21]: print(prime_eratosthenes(100))

2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
53
59
```

Fuente: <https://www.w3resource.com/python-exercises/list/python-data-type-list-exercise-34.php>