Universidad Nacional Autónoma de México

Facultad de Ciencias

Temas Selectos de Física Computacional 2

Ríos García Juan Saidt

Ejercicio 1 (1 punto):

En una celda Markdown contesta:

¿Qué es una palabra reservada en python? Enlista 5 ejemplos de nombres reservados en python y describe su función. Coloca tres ejemplos de nombres de variables que no son válidos en python. Explica por qué no son válidos.

Una palabra reserva es aquella que el lenguaje de programación reserva para una cierta función predeterminada y especifica del programa.

Algunos ejemplos son:

class: Se utiliza para comenzar la definición de una clase.

def: Se utiliza para comenzar la definición una función.

for: se utiliza para comenzar la definición de un ciclo for.

Algunos nombres de variables no validos son los siguientes:

$2023juan$ # No es valido ya que comienza con un numero.

$juan\ 2023$ # No es valido ya que contiene un espacio en blanco.

$class$ # No es valido ya que class es una palabra reservada

Ejercicio 2 (1 punto):

En una celda de código, escribe una función que encuentre las raíces de cualquier polinomio de segundo grado (implementar la chicharronera).

En una celda markdown detalla posibles casos en la cual la función podría fallar.

Usa dicha función para encontrar las raíces del polinomio $f(x) = 10x^2 - 2x$ e imprimelas en pantalla.

In [1]:
```python
import numpy as np
```

In [2]:
```python
# Definimos la función chicharronera
def chicharronera(a, b, c):
    x_1 = ((-b)+np.sqrt((b**2)-4*a*c))/(2*a)

    x_2 = ((-b)-np.sqrt((b**2)-4*a*c))/(2*a)

    return x_1, x_2
```

In [3]:
```python
# Definimos los coeficientes a, b, c
a = 10
b = -2
c = 0

# Obtenemos las raices llamando a la función
raiz_1, raiz_2 = chicharronera(a, b, c)

# Imprimimos las raices
print('Las soluciones son: {}, {}'.format(raiz_1, raiz_2))
```

```
Las soluciones son: 0.2, 0.0
```

La función fallara cuando tengamos una ecuación de segundo grado que sus raices sean complejas, como es el caso de la ecuación:

$$x^2 = -1$$

In [4]:
```python
# Error para raices complejas
# Definimos los coeficientes a, b, c
a = 1
b = 0
c = 1

# Obtenemos las raices llamando a la función
raiz_1, raiz_2 = chicharronera(a, b, c)

# Imprimimos las raices
print('Las soluciones son: {}, {}'.format(raiz_1, raiz_2))
```

```
Las soluciones son: nan, nan
C:\Users\juans\AppData\Local\Temp\ipykernel_5212\4239945788.py:3: RuntimeWarning: invalid value encountered in sqrt
  x_1 = ((-b)+np.sqrt((b**2)-4*a*c))/(2*a)
C:\Users\juans\AppData\Local\Temp\ipykernel_5212\4239945788.py:5: RuntimeWarning: invalid value encountered in sqrt
  x_2 = ((-b)-np.sqrt((b**2)-4*a*c))/(2*a)
```

Ejercicio 3 (1 punto):

Caida libre: El gran Galileo Galilei subió la torre de Pisa para determinar el tiempo que tarda un objeto en caer desde una altura H al suelo. Para ello, colocó un objeto de masa $m = 100kg$ en la torre y lo dejó caer. El tiempo que tardó en caer fue $t = 0.05616667$ minutos.

Considerando que la aceleración de la gravedad es $g = 9.8\frac{m}{s^2}$, escribe un programa que determine la altura de la torre de Pisa. El programa debe imprimir la altura en la pantalla.

Recuerda que el movimiento de caída libre simplificado cumple la ecuación: $y(t) = H + v_i t - \frac{1}{2}gt^2$.

In [5]:
```python
# Definimos la función altura_inicial
# Los parametros seran: t = tiempo, h_f = altura final, v_i = velocidad inicial, a = aceleracion
def altura_inicial(t, h_f, v_i, a):
    H = h_f - v_i*t + (1/2)*a*(t**2)

    return H
```

In [6]:
```python
# Definimos los parametros
t = 0.05616667 * 60 # convertimos a segundos
h_f = 0 # el marco de referencia tiene origen en el suelo
v_i = 0 # por ser caida libre
a = 9.8 # aceleracion de la gravedad

# Obtenemos la altura llamando a la función
altura_pisa = altura_inicial(t, h_f, v_i, a)

print('La altura de la torre de Pisa es: {} m.'.format(altura_pisa))
```

```
La altura de la torre de Pisa es: 55.64881660520021 m.
```

Ejercicio 4 (1 punto):

Crea una lista de números enteros del 0 al 99.

Escribe un programa que imprima en la pantalla una la lista que cumpla con las siguientes condiciones:

Los ultimos 10 elementos: debe imprimir lo siguiente [90, 91, 92, 93, 94, 95, 96, 97, 98, 99].

Los primeros 11 elementos: debe imprimir lo siguiente [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10].

La serie de elementos de la lista que están entre 60 y 75: debe imprimir lo siguiente [60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75].

El número 50: debe imprimir lo siguiente 50.

Debes aplicar el concepto de slicing para resolver este ejercicio (notación de puntos [:]).

Puntos menos si se crean listas nuevas para cada caso y/o se seleccionan manualmente los valores.

In [7]:
```python
# Creamos la lista de 0 a 99 por medio de las funciones list() y range()
lista = list(range(100))

print(lista)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41,
42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 8
1, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99]
```

In [8]:
```python
# Ultimos 10 elementos
print(lista[-10:])
```

```
[90, 91, 92, 93, 94, 95, 96, 97, 98, 99]
```

In [9]:
```python
# Primeros 11 elementos
print(lista[:11])
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

In [10]:
```python
# Elementos entre 60 y 75
print(lista[60:76])
```

```
[60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75]
```

In [11]:
```python
# Numero 50
print(lista[50])
```

```
50
```

Ejercicio 5 (1 punto):

Crea una lista de números enteros del 0 al 1000. A partir de dicha lista:

Guarda en una variable una lista con los números pares usando ciclos y condicionales. Imprime los últimos 10 elementos de la lista final.

Guarda en una variable una lista con los números impares usando listas comprensivas (comprehension list). Imprime los primeros 10 elementos de la lista final.

In [12]:
```python
# Creamos la lista
lista = list(range(1001))
```

In [13]:
```python
# Lista de pares
pares = []
for num in lista:
    if num % 2 == 0:
        pares.append(num)

print(pares[-10:])
```

```
[982, 984, 986, 988, 990, 992, 994, 996, 998, 1000]
```

In [14]:
```python
# Lista impares
impares = [num for num in lista if num % 2 != 0]

print(impares[:10])
```

```
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
```

Ejercicio 6 (2 puntos): Aproximación de π

Crea una función para aproximar π utilizando la serie de Leibniz. Usa $n = 100000$ e imprime el valor de tu aproximación, tanto el valor de $\frac{\pi}{4}$ como de π.

In [15]:
```python
# Definimos la serie de Leibniz
def serie_Leibniz(n):
    suma = 0
    for i in range(n):
        suma += ((-1)**i) / (2*i + 1)

    return suma
```

In [16]:
```python
# Calculamos las aproxmaciones para n = 100000
n = 100000
pi_4 = serie_Leibniz(n)
pi = pi_4 * 4

print('El valor de pi/4 es: {}'.format(pi_4))

print('El valor de pi es: {}'.format(pi))
```

```
El valor de pi/4 es: 0.7853956633974299
El valor de pi es: 3.1415826535897198
```

Ejercicio 7 (1 puntos): Identificador de vocales

Escribe una función que reciba una cadena, por ejemplo 'el pErro de la tiA chOnita' y regrese la cantidad de vocales que contiene. La función debe contener varios casos de prueba, if, elif y else. Con la cadena de ejemplo, imprime en pantalla el resultado de la función (10).

Para conocer si un caracter es una vocal, primero debes descomponer la cadena en caracteres individuales. Para ello observa el siguiente código:

In [17]:
```python
# Divide una cadena en caracteres individuales
cadena_prueba = 'el pErro de la tiA chOnita'
caracteres_individuales = list(cadena_prueba)
print(caracteres_individuales)
```

```
['e', 'l', ' ', 'p', 'E', 'r', 'r', 'o', ' ', 'd', 'e', ' ', 'l', 'a', ' ', 't', 'i', 'A', ' ', 'c', 'h', 'O', 'n', 'i', 't', 'a']
```

In [18]:
```python
# Utilizamos el metodo lower() para poner todo en minusculas
cadena = 'el pErro de la tiA chOnita'.lower()
caracteres = list(cadena)

print(caracteres)
```

```
['e', 'l', ' ', 'p', 'e', 'r', 'r', 'o', ' ', 'd', 'e', ' ', 'l', 'a', ' ', 't', 'i', 'a', ' ', 'c', 'h', 'o', 'n', 'i', 't', 'a']
```

In [19]:
```python
# Definimos un ciclo for para aplicar las pruebas a casa letra
num_vocales = 0
for letra in cadena:
    if letra == 'a':
        num_vocales += 1
    elif letra == 'e':
        num_vocales += 1
    elif letra == 'i':
        num_vocales += 1
    elif letra == 'o':
        num_vocales += 1
    elif letra == 'u':
        num_vocales += 1
    else:
        num_vocales += 0

print('El numero de vocales es: {}'.format(num_vocales))
```

```
El numero de vocales es: 10
```

In [20]:
```python
# Otra solucion es la siguiente:
vocales = ['a', 'e', 'i', 'o', 'u']

num_vocales = 0
for letra in cadena:
    if letra in vocales:
        num_vocales += 1

print('El numero de vocales es: {}'.format(num_vocales))
```

```
El numero de vocales es: 10
```

Ejercicio 8 (2 puntos):

Escribe una clase "calculadora" que regrese el resultado de las operaciones de suma, resta, multiplicacion, division, modulo y potencia. La clase debe contener un método para cada operación.

Los métodos que pudiesen presentar errores como division entre cero, deben ser manejados de tal manera que el programa no imprima ningun error (vimos una clausula para hacer esto).

Adicional a esto, los métodos modulo y potencia deben ser generales, es decir, deben funcionar para cualquier número entero positivo (es un parámetro extra del método). Por ejemplo, la potencia de 2 elevado a 3 es 8, la potencia de 5 elevado a 3 es 125, etc.

Crea una instancia de la clase calculadora y prueba cada uno de los métodos, con valores de tu elección. Para los métodos que pudiesen presentar errores da dos ejemplos, uno totalmente funcional (3/5 por ejemplo) y otro que sea conflictivo pero que no regresa ningún error (4/0 por ejemplo).

In [21]:
```python
# Definimos la clase calculadora
class calculadora:
    def suma(a, b):
        suma_cal = 0
        try:
            suma_cal = a + b
            return suma_cal
        except TypeError:
            print('Los valores proporcionados no son adecuados')

    def resta(a, b):
        resta_cal = 0
        try:
            resta_cal = a - b
            return resta_cal
        except TypeError:
            print('Los valores proporcionados no son adecuados')

    def multiplicacion(a, b):
        mult_cal = 0
        try:
            mult_cal = a * b
            return mult_cal
        except TypeError:
            print('Los valores proporcionados no son adecuados')

    def division(a,b):
        div_cal = 0
        try:
            div_cal = a / b
            return div_cal
        except ZeroDivisionError:
            print('Los valores proporcionados no son adecuados')
        except TypeError:
            print('Los valores proporcionados no son adecuados')

    def modulo(a, b):
        mod_cal = 0
        try:
            mod_cal = a % b
            return mod_cal
        except ZeroDivisionError:
            print('Los valores proporcionados no son adecuados')
        except TypeError:
            print('Los valores proporcionados no son adecuados')

    def potencia(a, b):
        pot_cal = 0
        try:
            pot_cal = a**b
            return pot_cal
        except TypeError:
            print('Los valores proporcionados no son adecuados')
```

In [22]:
```python
# Prueba del metodo suma()
a = 5
b = 7

suma = calculadora.suma(a, b)

print(suma)
```

```
12
```

In [23]:
```python
# Prueba del metodo resta()
a = 5
b = 7

resta = calculadora.resta(a, b)

print(resta)
```

```
-2
```

In [24]:
```python
# Prueba del metodo mult()
a = 5
b = 7

mult = calculadora.multiplicacion(a, b)

print(mult)
```

```
35
```

In [25]:
```python
# Prueba del metodo div()
a = 5
b = 7

div = calculadora.division(a, b)

print(div)
```

```
0.7142857142857143
```

In [26]:
```python
# Prueba del metodo div()
a = 5
b = 0

div = calculadora.division(a, b)

print(div)
```

```
Los valores proporcionados no son adecuados
None
```

In [27]:
```python
# Prueba del metodo modulo()
a = 19
b = 2
mod = calculadora.modulo(a, b)

print(mod)
```

```
1
```

In [28]:
```python
# Prueba del metodo modulo()
a = 19
b = 0
mod = calculadora.modulo(a, b)

print(mod)
```

```
Los valores proporcionados no son adecuados
None
```

In [29]:
```python
# Prueba del metodo potencia()
a = 2
b = 8
pot = calculadora.potencia(a, b)

print(pot)
```

```
256
```

In [ ]:
```python

```