Escalona_Joaquin_3

August 25, 2018

Hola, pueden encontrar este documento y los códigos aquí copiados para que puedan ejecutarlos y evaluarlos con mayor facilidad pinchando aquí :) GitHub

1 Ejercicio 1

Debe escribir un programa que le pida a un usuario adivinar un nombre, pero sólo tienen 3 posibilidades hasta que el programa se cierra.

1.1 Solucion

Elegir aleatoriamente --> De aquí saqué la forma para elegir aleatoriamente un elemento de una lista.

El programa a continuación elegirá aleatoriamente un elemento de la lista y el usuario debe adivinarlo. Si no lo logra, el programa imprimirá el nombre que ha elegido aleatoriamente.

```
In [ ]: #IMPORTAR RANDOM PARA ELEGIR UN ELEMENTO AL AZAR
        import random
        nombres = ['Amelia','Luis','Jaime','Neil','Constanza','Joaquin','Yuuki']
        nom_elegido=random.choice(nombres)
        #IMPRIMIR CONDICIONES DEL JUEGO
        print('------ ADIVINA EL NOMBRE ------')
        print('\n Tendras solo 3 oportunidades')
        print('Los nombres a elegir son = ')
        for i in nombres:
            print '*',i
        #PEDIR UN NOMBRE
        n=raw_input('\n Comencemos, elige un nombre = ')
        #CONTADOR (POSIBILIDADES) = 1
        count = 1
        #CICLO CON TOPE = 3
        while True:
        #SI EL USUARIO ADIVINA
            if n == nom_elegido:
                print ('Correcto! has adivinado :)')
                break
        #SINO, SE DESCUENTA UNA POSIBILIDAD (AGREGANDOLE A COUNT)
            else:
```

```
            count+=1
        #NONE NECESARIO PARA QUE EL USUARIO VUELVA A INTENTAR
            n=None
            n=raw_input('\n Nope, intentalo nuevamente,elige un nombre = ')
        #SI LLEGA A 3 POSIBILIDADES
            if count == 3:
                print ('Has ocupado tus 3 opciones')
                print 'El nombre era = ',nom_elegido
                break
```

2 Ejercicio 2

Arreglen el código adjunto para que haga lo esperado: solo deje "pasar" a personal con uno de los tres nombres especificados. Deben usar como mucho solo una instancia de "==".

Codigo adjunto:

```
import sys
print("Hello. Please enter your name:")
name = sys.stdin.readline().strip()
if name == "Ana" or "Maria" or Itziar":
    print("Access granted.")
else:
    print("Access denied.")
```

2.1 Solución

La forma en que se me ha ocurrido hacer el problema es incluir los nombres del personal a una lista y hacer un ciclo IF como una puerta de entrada: si el nombre ingresado se encuentra en la lista, se abre la puerta, sino, no.

```
In [ ]: import sys
        #LISTA CON NOMBRES PERMITIDOS
        personal= ['Ana','Maria','Itziar']
        print("Hello. Please enter your name:")
        name = sys.stdin.readline().strip()
        #SI EL NOMBRE SE ENCUENTRA EN LA LISTA, PERMITIR
        if name in personal:
            print('Access granted.')
        #NO PERMITIR
        else:
            print('Acces denied.')
```

3 Ejercicio 3

Sea x un número entero.

A) Describa un algoritmo a base de iteraciones para comprobar si x es un número primo.

B) Escriba un programa en Python que solicite un input de un número entero, y que usa el algoritmo de arriba para comprobar si este número es un número primo.

C) Escriba un programa para comprobar si un número es el cuadrado de un número primo. Osea, que la raíz del número ingresado sea un número primo.

Nota: Pruebe al inicio si el número es un cuadrado de un número entero. Si eso es verdad, pruebe también si este número es primo.

3.1 Solución apartado A)

La función **if any(lista)** la aprendí en la sección "Mas tipos" >> Funciones Utiles del curso que da SóloLearn. Para que un número sea primo, éste debe ser divisible sólo por 1 y por sí mismo. Por lo tanto, si existiera algun i (if any) con el cual el cuociente entre el numero ingresado x e i sea igual a 0, el número deja de ser primo.

Nota: expliqué el desarrollo del programa en el enunciado para evitar comentarios en el programa y que quedara feito.

```
In [ ]: #APARTADO A)
        #-------------------------
        #DEFINIMOS FUNCION QUE ARROJA SI UN VALOR ES PRIMO O NO
        #INPUT = X
        #SI X ES PRIMO, RETORNA 1
        #SI NO, RETORNA 0
        def es_primo(x):
        #INICIO LOOP DESDE 2 HASTA X-1
            for i in range(2,x):
                if any([x%i==0]):
                    return 0
            return 1
```

3.2 Solución apartado B)

Esto va junto con el programa anterior (apartado A)

```
In [ ]: x = int(input('Ingresa un numero y dire si es primo o no = '))
        if x==1:
            print('No es primo :( ')
        elif x>0:
            if es_primo(x) == 1:
                print('Es primo!')
            else:
                print('No es primo :( ')
        else:
            print('Debe ser un numero entero y positivo!')
```

3.3 Solución apartado C)

Esta parte del código va junto con el apartado B (no necesario) y A (necesario). Se importa la función sqrt del módulo math para poder realizar la operación raíz :)

```
In [ ]: y = int(input('Ingresa un numero y dire si es el cuadrado de un primo = '))
        #SI NUMERO INGRESADO ES MAYOR A 0, SE SACA LA RAIZ DEL NUMERO
        if y>0:
            raiz=int(sqrt(y))
        #SINO, PEDIR QUE SEA POSITIVO
        else:
            print('Debe ser positivo')
        #1 POR CONVENIO, NO ES CONSIDERADO PRIMO
        #LO LEI AQUI: https://es.wikipedia.org/wiki/N%C3%BAmero_primo
        if raiz==1:
            print('No lo es :( ')
        #SI LA RAIZ ES DISTINTO DE 1
        else:
        #SI LA RAIZ ES PRIMO (VER APARTADO A)
            if es_primo(raiz) == 1:
                print 'Lo es! del primo ',raiz
            else:
                print('No lo es :( ')
```

4 Ejercicio 4:

En el siguiente ejercicio, escribiremos programas que usan algoritmos diferentes para calcular la tercera raíz con una precisión de 0.01.

a) Escriba un programa que use un algoritmo a base de exhaustive enumeration para la determinar la tercera raíz. £Cuántas iteraciones necesita para determinarla con la precisión deseada para los números 25, 500 y 10000?

b) Escriba un programa que calcula la tercera raíz con el algoritmo de bisección. £Cuántas iteraciones necesita para determinarla con la precisión deseada para los números 25, 500 y 10000?

c) Escriba un programa que calcula la tercera raíz con el método de Newton. £Cuantas iteraciones necesita para determinarla con la precisión deseada para los números 25, 500 y 10000?

4.1 Solución apartado A)

Importante! Soluciones A Y B las tuve que copiar descaradamente (con leves modificaciones) ya que nisiquiera sabía en qué consistían los métodos de programación pedidos. Los pasos matemáticos lo explica la profesora de forma muy clara.

La solución original de este apartado la puedes encontrar en la página 68 de la lecture3-4.pdf .

```
In [ ]: #EXHAUSTIVE ENUMERATION
        def raiz_cubica(x):
        #PRECISION
            epsilon = 0.01
            step = epsilon**3
            ans = 0.0
        #CONTADORA DE ITERACIONES
            count=0

            while abs(ans**3 - x) >= epsilon and ans <=x:
        #AGREGAR ITERACION HASTA QUE SE ROMPA EL CICLO
                count+=1
                ans += step
            if abs(ans**3 - x) >= epsilon:
                print 'No hemos encontrado la raiz de ',x

            print '* La raiz de', x, 'es aproximadamente', ans
            print '* Ocurrieron',count,'iteraciones \n'

        raiz_cubica(25)
        raiz_cubica(500)
        raiz_cubica(10000)
```

El programa arroja este resultado:

```
* La raiz de 25 es aproximadamente 2.92362800005.
* Ocurrieron 2923628 iteraciones

* La raiz de 500 es aproximadamente 7.93695300076
* Ocurrieron 7936953 iteraciones

* La raiz de 10000 es aproximadamente 21.5443400005
* Ocurrieron 21544340 iteraciones
```

4.2 Solución apartado B)

De aquí pude entender mas o menos el algoritmo planteado por la profesora. Me parece bastante interesante este método de programación.

```
In [ ]: #BISECCION
        #DESCARADAMENTE COPIADO DEL MATERIAL ENVIADO POR PROFESORA
        #PUEDES ENCONTRAR EL ORIGINAL EN LECTURE3-4, PAG 81

        #SE DEFINE FUNCION RAIZ CUBICA
        def raiz_cubica(x):
        #PRECISION
            epsilon = 0.01
        #MINIMO Y MAXIMO DEL INTERVALO
            low,high = 0.0, max(1.0,x)
```

```
        #RESPUESTA
            ans = (high + low)/2.0
        #CONTADORA DE ITERACIONES
            count = 0

            while abs(ans**3 - x) >= epsilon:
                #print 'low =',low, 'high =',high, 'ans = ',ans
                count +=1
        #SI EL CUADRADO DE LA POSIBLE RESPUESTA ES MENOR QUE X
        #ENTONCES DEBE ESTAR A LA IZQUIERDA
                if ans**3 < x:
                    low = ans
        #SI ES MAYOR, DEBE ESTAR A LA DERECHA
                else:
                    high = ans

                ans = (high + low)/2.0
            print '* La raiz de', x, 'es aproximadamente', ans
            print '* Ocurrieron',count,'iteraciones \n'
        raiz_cubica(25)
        raiz_cubica(500)
        raiz_cubica(10000)
```

El programa arroja este resultado:

```
* La raiz de 25 es aproximadamente 2.92434692383
* Ocurrieron 14 iteraciones

* La raiz de 500 es aproximadamente 7.93695449829
* Ocurrieron 19 iteraciones

* La raiz de 10000 es aproximadamente 21.5443409979
* Ocurrieron 28 iteraciones
```

4.3 Solución apartado C)

```
In [ ]: #NEWTON
        #PUEDES ENCONTRAR EL ORIGINAL EN LECTURE3-4, PAG 89
        #SE DEFINE FUNCION RAIZ CUBICA
        def raiz_cubica(x):
        #PRECISION
            epsilon = 0.01
        #RESPUEST
            guess = x/2.0
        #CONTADORA DE ITERACIONES
            count=0
            while abs(guess**3 - x) >= epsilon:
                count+= 1
```

```
        #METODO DE NEWTON, DENOMINADOR ES LA DERIVADA DE LA FUNCION
                guess = guess - (((guess**3) - x) / (3*(guess**2)))
            #print('* Ocurrieron '+str(count)+ ' iteraciones \n')
            print '* La raiz de', x, 'es aproximadamente', guess
            print '* Ocurrieron',count,'iteraciones \n'
        raiz_cubica(25)
        raiz_cubica(500)
        raiz_cubica(10000)
```

El programa arroja este resultado:

```
* La raiz de 25 es aproximadamente 2.9242328368
* Ocurrieron 6 iteraciones

* La raiz de 500 es aproximadamente 7.93700527704
* Ocurrieron 12 iteraciones

* La raiz de 10000 es aproximadamente 21.5443469166
* Ocurrieron 17 iteraciones
```

5 Ejercicio 5

Ejercicio 5: Usted tiene las siguientes ecuaciones:

a) $x^2 = 4x$

b) $e^x = 4x$

c) $10x = x^2$

Para cada ecuación, defina una función $f(x)$ de forma que el cero de la función f sea la solución de la ecuación. Después, calcule también df/dx y use el método de Newton para determinar la solución. **(No nos interesa la solución trivial x = 0 en caso de a) y c). Si resulta zero, cambie el supuesto inicial.)**

5.1 Solución apartado A)

Ésta solución y las que siguen, han sido basadas tras leer la entrada de *Shah* en este link

```
In [ ]: #definimos funciones

        # f1(x)---> x^2 - 4x
        # f1'(x)---> 2x - 4
        def f1(x):
            return x**2 - 4*x

        def df1(x):
            return 2*x-4
```

```
        #supuesto inicial (con 2.0 retorna ZeroDivisionError)
        xnew = [3.0]
        #error tolerado al aproximar
        erro = 0.001
        #raiz de la funcion
        resp = xnew[-1]
        print '-------- Metodo de Newton --------'
        print 'Funcion = x^2 - 4x'
        #try/except por si ocurre division por 0
        try:
            while True:
                print 'Mejor aproximacion = ',resp
        #metodo de newton-raphson
                resp = resp - (f1(resp)/df1(resp))
        #agregar esta resp a xnew
                xnew.append(resp)
        #alcanzar el error exigido
                if abs(xnew[-2]-xnew[-1]) <= erro:
                    break
        except (ZeroDivisionError):
            print 'Ha ocurrido una division por 0. No se puede continuar'
        print '***La aproximacion final es',resp
```

El programa arroja:

```
-------- Metodo de Newton --------
Funcion = x^2 - 4x
Mejor aproximacion =  3.0
Mejor aproximacion =  4.5
Mejor aproximacion =  4.05
Mejor aproximacion =  4.0006097561
***La aproximacion final es 4.00000009292
```

5.2 Solución apartado B)

```
In [ ]: #para la funcion exponencial
        from math import exp
        # definimos funciones
        # f2(x) ---> e^x - 4x
        # df2(x) ---> e^x - 4
        def f2(x):
            return exp(x) - 4*x
        def df2(x):
            return exp(x) - 4

        #supuesto inicial
        xnew = [3.0]
        #error tolerado al aproximar
```

```
erro = 0.001
#raiz de la funcion
resp = xnew[-1]
print '-------- Metodo de Newton --------'
print 'Funcion = e^x - 4x'
#try/except por si ocurre division por 0
try:
    while True:
        print 'Mejor aproximacion = ',resp
#metodo de newton-raphson
        resp = resp - (f2(resp)/df2(resp))
#agregar esta resp a xnew
        xnew.append(resp)
#alvanzar el error exigido
        if abs(xnew[-2]-xnew[-1]) <= erro:
            break
except (ZeroDivisionError):
    print 'Ha ocurrido una division por 0. No se puede continuar'
print '***La aproximacion final es',resp
```

El programa arroja:

```
-------- Metodo de Newton --------
Funcion = e^x - 4x
Mejor aproximacion =  3.0
Mejor aproximacion =  2.49734118533
Mejor aproximacion =  2.23221940087
Mejor aproximacion =  2.15860801401
Mejor aproximacion =  2.15331857522
***La aproximacion final es 2.15329236475
```

5.3 Solución apartado C)

```
In [ ]: #definimos funciones

        #f3(x)--->10x - x^2
        #df3(x)--->10 - 2x
        def f3(x):
            return 10*x - x**2
        def df3(x):
            return 10 - 2*x

        #supuesto inicial
        xnew = [7.0]
        #error tolerado al aproximar
        erro = 0.001
        #raiz de la funcion
        resp = xnew[-1]
```

```
print '-------- Metodo de Newton --------'
print 'Funcion = 10 - x^2'
#try/except por si ocurre division por 0
try:
    while True:
        print 'Mejor aproximacion = ',resp
#metodo de newton-raphson
        resp = resp - (f3(resp)/df3(resp))
#agregar esta resp a xnew
        xnew.append(resp)
#alcanzar el error exigido
        if abs(xnew[-2]-xnew[-1]) <= erro:
            break
except (ZeroDivisionError):
    print 'Ha ocurrido una division por 0. No se puede continuar'
print '***La aproximacion final es',resp
```

El programa arroja:

```
-------- Metodo de Newton --------
Funcion = 10 - x^2
Mejor aproximacion =  7.0
Mejor aproximacion =  12.25
Mejor aproximacion =  10.349137931
Mejor aproximacion =  10.0113941065
Mejor aproximacion =  10.000012953
***La aproximacion final es 10.0
```