



INS-203

Lógica de programación

Nombre:

Daniel Alonso - 1066734

Lía Báez - 1069360

Julio Beras - 1070044

Edgar Félix - 1070185

Profesor:

Lorenzo Martínez

Proyecto final: Borrador 1

Introducción:

En el siguiente documento pretendemos presentar un reporte de las ideas y construcciones que tenemos para el desarrollo de una aplicación con Python 3 que pretende simplificar el proceso de calcular y graficar funciones para usuarios universitarios.

Los integrantes del proyecto, como estudiantes de matemática, queremos simplificar los procedimientos para graficar funciones y realizar cálculos algebraicos, dándole al usuario la mayor información posible del procedimiento realizado, de manera que el usuario pueda entender, lo más posible, lo que acaba de hacer, y que no reciba simplemente un resultado numérico o simbólico para el usuario.

Descripción:

El proyecto consiste en construir un programa que compile múltiples funciones matemáticas relacionadas con cálculo y geometría analítica, fundamentalmente centrado en funciones de dos y varias variables, con cálculos simbólicos como por ejemplo hallar las n derivadas de una función en dos o más variables, las n integrales de una función, graficar funciones, y presentar la mayor información posible de la función o el gráfico presentado.

El programa **inserte nombre creativo** tiene como objetivo ayudar al usuario a comprender de una manera más integra conceptos básicos del álgebra de una manera dinámica e interactiva, combinando los recursos de índole matemático que nos ofrecen las librerías de Python y los conocimientos en dicha ciencia que hemos adquirido como estudiantes. Se intenta crear un ambiente didáctico y amigable para el usuario, fácil de utilizar que provea la mayor cantidad de información posible sin procesos complicados, y que exponga los datos y resultados de una manera eficiente y práctica.

Algunas informaciones relevantes que se podrían presentar de un gráfico de una función de dos dimensiones pueden ser, por ejemplo:

- Puntos de inflexión
- Cortes de la función en los diferentes ejes
- Cortes de varias funciones entre sí, como solución a sistemas de ecuaciones no lineales y lineales

Análisis:

La mayoría de los programas para graficar funciones que son utilizados diariamente por los estudiantes de matemáticas simplemente les muestran un resultado final sin darles una explicación alguna de donde proviene toda la información.

Por esta razón, nos surge la idea de crear un programa un poco más didáctico, un programa que pueda brindar más información acerca de una función y explicar al usuario de una forma gráfica informaciones relevantes como son los puntos de inflexión, los interceptos de una función en el plano y los puntos en los que se cortan dos o más funciones distintas, que podrían ser vistos como las soluciones para sistemas de ecuaciones lineales y no lineales.

De esta manera, cualquier estudiante o aficionado de las matemáticas, que le resulte más difícil comprender conceptos de manera teórica podría asimilarlos al tener en frente una gráfica brindándole una gran cantidad de información.

El programa debe ser capaz de recibir e interpretar los datos introducidos por el usuario y de realizar todos los cálculos necesarios para devolver la información en forma gráfica previamente solicitada por el usuario sobre alguna función. También, debe ser un software eficiente, que brinde las respuestas en el menor tiempo posible, que sea fácil de mantener, seguro y escalable.

Diseño:

La aplicación se construirá en Python 3, utilizando librerías diversas como NumPy, SciPy, Matplotlib y SymPy.

El programa tendrá dividida en diferentes bloques definiendo funciones, cada una de las funcionalidades del programa, utilizando la menor cantidad de recursos de memoria posibles para la más rápida ejecución y eficiencia del programa.

Luego de construir todas las funciones del programa, el programa tendrá un menú principal, donde se verifica que cada dato entrado por el usuario es correcto, así las funciones pueden continuar normalmente, de manera que el usuario no obtenga un error genérico de Python que este no pueda solucionar.

Todos los bloques del programa estarán construidos como funciones, utilizando def o lambda para facilitar el proceso de construcción del programa, poder identificar errores de una manera más eficiente, y poder agregar, de una manera más organizada, la mayor cantidad de funciones posibles, llamándolas desde el menú principal al ser las mismas accionadas por el usuario.

Al Python ser un lenguaje de código abierto, el código también será abierto a ser utilizado y modificado por cualquier usuario, de manera que este pueda modificar cualquier elemento del algoritmo y ajustarlo a sus necesidades, y, de la misma manera, trataremos de ofrecer la mayor

funcionalidad posible.

Código fuente del programa:

Aquí podemos ver el código de la aplicación:

```
# -*- coding: utf-8 -*-
"""
Created on Sun Jun 18 11:35:04 2017

@author: dreth
"""

# estos bloques try chequean si el usuario tiene las librerías necesarias para correr el programa
# en el caso contrario, cierran el programa inmediatamente y avisa al usuario que librería falta
try:
    from sympy import simplify
except ModuleNotFoundError:
    try:
        from sys import exit as ex
        ex('Error!')
    except SystemExit:
        print('\n'           Necesita instalar la librería Sympy para poder utilizar la
aplicación          \n')

try:
    from numpy import shape
except ModuleNotFoundError:
    try:
        from sys import exit as ex
        ex('Error!')
    except SystemExit:
        print('\n'           Necesita instalar la librería Numpy para poder utilizar la
aplicación          \n')

try:
    from matplotlib import pyplot
except ModuleNotFoundError:
    try:
        from sys import exit as ex
        ex('Error!')
    except SystemExit:
        print('\n'           Necesita instalar la librería matplotlib para poder utilizar la
aplicación          \n')
```

```

try:
    del sympy.simplify
    del numpy.shape
    del matplotlib.plot
    del ex
except (NameError,ModuleNotFoundError,SyntaxError):
    print('\n')

# Recursos extra
# Esta funcion chequea si un número es un cuadrado perfecto para
# la funcion que calcula digitos de raices de cuadrados no perfectos
# Luego de chequear que no es cuadrado perfecto en aquella funcion
# calcula la cantidad de digitos, pero si decide que es cuadrado perfecto
# entonces modifica la cantidad de digitos decimales que va a calcular a 1 solo

def continuar():
    print('\n ¿Desea continuar? (y,n) \n')
    k = input('(y,n) : ')
    if k == 'y' or k == 'Y':
        main()
    else:
        return

def is_square(entero):
    from math import sqrt
    entero = int(entero)
    root = sqrt(entero)
    if int(root + 0.5) ** 2 == entero:
        return True
    else:
        return False

# Funciones del programa
def graf_funcion_cuadratica(x0,x1):
    # Aqui la funcion trata de convertir los valores a float para agarrar
    # errores de atributo, en caso de que el usuario introduzca datos del tipo incorrecto
    try:
        x0 = float(x0)
        x1 = float(x1)
    except (NameError,ValueError):
        print('\nDebe introducir valores numéricos!')
        return
    from numpy import linspace as lnp
    from matplotlib import pyplot as plt

```

```

x = lnp(x0,x1,num=1000)
print('f(x) = a(x^2) + b(x) + c')
# pide al usuario los coeficientes luego de imprimir el formato
# de la funcion que va a graficar
try:
    a = float(input('a = '))
    b = float(input('b = '))
    c = float(input('c = '))
    y = a*(x**2) + b*x + c
    # k imprime la funcion que el usuario ha introducido
    k = ('f(x) = ({0})(x^2) + ({1})(x) + ({2})'.format(a,b,c))
except (ValueError,NameError,AttributeError):
    print('\n Debe introducir valores numéricos!')
    return
# Se aplican las diferentes funciones para graficar con matplotlib
fig, ax = plt.subplots()
ax.plot(x,y)
plt.title('gráfico de {}'.format(k))
fig.canvas.set_window_title('gráfico de {}'.format(k))
ax.grid(True, which='both')
if (x0 <= 0 and x1 > 0) or (x0 > 0 and x1 <= 0):
    ax.spines['left'].set_position('zero')
    ax.spines['right'].set_position('zero')
plt.show()
# Pregunta al usuario si desea continuar, si dice que si, lo devuelve al menu principal
# si dice que no, la funcion se cierra y no devuelve ningun valor.
continuar()

def graf_funcion_cubica(x0,x1):
    try:
        x0 = float(x0)
        x1 = float(x1)
    except (NameError,ValueError):
        print('\nDebe introducir valores numéricos!')
        return
    from numpy import linspace as lnp
    from matplotlib import pyplot as plt
    x = lnp(x0,x1,num=1000)
    print('f(x) = a(x^3) + b(x^2) + c(x) + d')
    try:
        a = float(input('a = '))
        b = float(input('b = '))
        c = float(input('c = '))
        d = float(input('d = '))
        y = a*(x**3) + b*(x**2) + c*x + d
    
```

```

k = ('f(x) = ({0})(x^3) + ({1})(x^2) + ({2})(x) + ({3})'.format(a,b,c,d))
except (ValueError,NameError,AttributeError):
    print('\n Debe introducir valores numéricos \n')
# graficando
fig, ax = plt.subplots()
ax.plot(x,y)
plt.title('gráfico de {0}'.format(k))
fig.canvas.set_window_title('gráfico de {0}'.format(k))
ax.grid(True, which='both')
if (x0 <= 0 and x1 > 0) or (x0 > 0 and x1 <= 0):
    ax.spines['left'].set_position('zero')
    ax.spines['right'].set_position('zero')
plt.show()
continuar()

def graf_funcion_lineal(x0,x1,m,b):
    # input check
    try:
        x0 = float(x0)
        x1 = float(x1)
    except (NameError,ValueError):
        print('\nDebe introducir valores numéricos!')
        return
    import numpy as np
    from matplotlib import pyplot as plt
    x = np.linspace(x0,x1,num=1000)
    y = m * x + b
    k = ('f(x) = ({0})(x) + ({1})'.format(m,b))
    # graficando
    fig, ax = plt.subplots()
    ax.plot(x,y)
    plt.title('gráfico de {0}'.format(k))
    fig.canvas.set_window_title('gráfico de {0}'.format(k))
    ax.grid(True, which='both')
    ax.spines['left'].set_position('zero')
    ax.spines['right'].set_position('zero')
    plt.show()
    continuar()

def graf_mov_armonico(t0,t1,cos_o_sen):
    cond = [(cos_o_sen == 'sin' or cos_o_sen == 'sen' or cos_o_sen == 'Sen' or cos_o_sen ==
'Sin'),(cos_o_sen == 'cos' or cos_o_sen == 'Cos')]
    try:
        t0, t1 = float(t0), float(t1)
    except (NameError,ValueError):

```

```

print('\nDebe introducir valores numéricos!')
return
import numpy as np
from matplotlib import pyplot as plt
t = np.linspace(t0,t1,num=1000)
if cond[0]:
    print('\nf(x) = (A) * sen(ωt - φ)')
elif cond[1]:
    print('\nf(x) = (A) * cos(ωt - φ)')
else:
    print('\n Error: debe elegir el tipo de función para el movimiento armónico! (cos,sen) \n')
try:
    A = float(input('Amplitud = '))
    ω = float(input('Frecuencia angular = '))
    φ = float(input('Constante de fase = '))
    if cond[0]:
        y = (A)*np.sin(ω*t-φ)
        if φ < 0:
            φ = (-1)*φ
        k = ('f(x) = ({0}) * sen({1}t + ({2}))'.format(A,ω,φ))
        φ = (-1)*φ
    elif φ > 0:
        k = ('f(x) = ({0}) * sen({1}t - ({2}))'.format(A,ω,φ))
    else:
        k = ('f(x) = ({0}) * sen({1}t)'.format(A,ω))
    elif cond[1]:
        y = (A)*np.cos(ω*t-φ)
        if φ < 0:
            φ = (-1)*φ
        k = ('f(x) = ({0}) * cos({1}t + ({2}))'.format(A,ω,φ))
        φ = (-1)*φ
    elif φ > 0:
        k = ('f(x) = ({0}) * cos({1}t - ({2}))'.format(A,ω,φ))
    else:
        k = ('f(x) = ({0}) * cos({1}t)'.format(A,ω))
    print(k)
except:
    print('\n Error: debe elegir el tipo de función para el movimiento armónico! (cos,sen) \n')
except (NameError,ValueError):
    print('\n Debe introducir valores numéricos \n')
return
# graficando
fig, ax = plt.subplots()
ax.plot(t,y)
plt.title('gráfico de {}'.format(k))
fig.canvas.set_window_title('gráfico de {}'.format(k))

```

```

ax.grid(True, which='both')
if (t0 <= 0 and t1 > 0) or (t0 > 0 and t1 <= 0):
    ax.spines['left'].set_position('zero')
    ax.spines['right'].set_position('zero')
plt.show()
continuar()

def graf_funcion_val(x0,x1,y):
    # input check
    try:
        x0 = float(x0)
        x1 = float(x1)
    except (NameError,ValueError):
        print('\nDebe introducir valores numéricos!')
        return
    # importa numpy como p para facilitar la sintaxis en la entrada de la función (y)
    import numpy as p
    from matplotlib import pyplot as plt
    x = p.linspace(x0,x1,num=1000)
    k = ('f(x) = {0}'.format(y))
    y = eval(y)
    fig, ax = plt.subplots()
    ax.plot(x,y)
    plt.title('grafico de {0}'.format(k))
    fig.canvas.set_window_title('grafico de {0}'.format(k))
    ax.grid(True, which='both')
    if (x0 <= 0 and x1 > 0) or (x0 > 0 and x1 <= 0):
        ax.spines['left'].set_position('zero')
        ax.spines['right'].set_position('zero')
    plt.show()
    continuar()

def digitos_e(n):
    try:
        int(n)
    except (ValueError,NameError):
        print('\n ¡Debe introducir valores enteros! \n')
    n += 1
    if n > 7e4:
        print('\n ¿El proceso puede tardar mucho, seguro que desea continuar?')
        k = input('\n ¿Desea continuar? (y,n) : ')
        if k == 'y' or k == 'Y':
            pass
        elif k == 'n' or k == 'N':
            print('\n Operación abortada')

```

```

        return
    else:
        print('\n Error')
        return
import sympy as sy
return sy.N(sy.E, n)

def digitos_pi(n):
    try:
        int(n)
    except (ValueError,NameError):
        print('\n ¡Debe introducir valores enteros! \n')
    n += 1
    if n > 7e4:
        print('\n¿El proceso puede tardar mucho, seguro que desea continuar?')
        k = input('\n ¿Desea continuar? (y,n) : ')
        if k == 'y' or k == 'Y':
            pass
        elif k == 'n' or k == 'N':
            print('\n Operación abortada')
            return
        else:
            print('\n Error')
            return
    import sympy as sy
    return sy.N(sy.pi, n)

def digitos_sqrt_irr(a,n):
    try:
        int(n)
    except (ValueError,NameError):
        print('\n ¡Debe introducir valores enteros para la cantidad de dígitos! \n')
    try:
        float(a)
    except (ValueError,NameError):
        print('\n ¡Debe introducir valores numéricos para la raíz que desea evaluar! \n')
    if is_square(a) == True:
        n = 2
    else:
        n += 1
    if n > 7e4:
        print('\n¿El proceso puede tardar mucho, seguro que desea continuar?')
        k = input('\n ¿Desea continuar? (y,n) : ')
        if k == 'y' or k == 'Y':

```

```

    pass
elif k == 'n' or k == 'N':
    print('\n Operación abortada')
    return
else:
    print('\n Error')
    return
import sympy as sy
return sy.N(sy.sqrt(a), n)

def resolver_ecuacion(ecuacion, variable_a_resolver):
    try:
        if '=' in ecuacion:
            print('\n La ecuación esta automáticamente igualada a cero, no agregue ninguna
igualdad en el campo de ecuación \n')
        if type(ecuacion) != str or type(variable_a_resolver) != str:
            print('\n Debe introducir parámetros del tipo cadena para el campo ecuación y carácter
para campo variable a resolver \n')
        import sympy as sy
        ecuacion = sy.simplify(ecuacion)
        var = sy.symbols('{0}'.format(variable_a_resolver))
        solucion = sy.solve(ecuacion, var)
        if (type(solucion) != list):
            print('\n La ecuación esta automáticamente igualada a cero, no agregue ninguna
igualdad en el campo de ecuación \n')
            return
        return solucion
    except NotImplementedError:
        print('\n La ecuación no puede ser resuelta por el programa \n intente una ecuación menos
intensa \n')
    except SyntaxError:
        print('\n Error de introducción de datos, inténtelo de nuevo \n')

def derivar_funcion(funcion, variable):
    if '=' in funcion:
        print('\n La ecuación esta automáticamente igualada a cero, no agregue ninguna igualdad
en el campo de ecuación \n')
    if type(funcion) != str or type(variable) != str:
        print('\n Debe introducir parámetros del tipo cadena para el campo ecuación y carácter
para campo variable a resolver \n')
    import sympy as sy
    funcion = sy.simplify(funcion)
    var = sy.symbols('{0}'.format(variable))
    deriv = sy.diff(funcion, var)

```

```

return deriv

def integrar_funcion(funcion,variable):
    if '=' in funcion:
        print('\n La ecuación esta automáticamente igualada a cero, no agregue ninguna igualdad
en el campo de ecuación \n')
    if type(funcion) != str or type(variable) != str:
        print('\n Debe introducir parámetros del tipo cadena para el campo ecuación y carácter
para campo variable a resolver \n')
    import sympy as sy
    funcion = sy.simplify(funcion)
    var = sy.symbols('{0}'.format(variable))
    integral = sy.integrate(funcion,var)
    return integral

def main():
    # menu principal
    print('._____._.')
    print('\n      Menú principal      \n\n ¿Qué función desea utilizar? \n')
    print('1 - Graficar')
    print('2 - Resolver ecuaciones')
    print('3 - Calcular dígitos de un número irracional')
    print('4 - Cálculos simbólicos')
    print('c - Cerrar/cancelar')
    k = input('\n Introduzca la opción: ')
    if k == '1':
        # menu de graficacion
        print('._____._.')
        print('\n      Menú de graficación ')
        print('\n ¿Qué función desea graficar? \n')
        print('1 - Función cuadrática')
        print('2 - Función cúbica')
        print('3 - Función lineal')
        print('4 - Gráfica de movimiento armónico')
        print('5 - Otra')
        print('m - volver al menú anterior')
        print('c - Cerrar/cancelar \n')
        k = input('\n Introduzca una opción: ')

    if k == '1':
        x0 = input('Introduzca el límite inferior del intervalo a evaluar (x0): ')
        x1 = input('Introduzca el límite superior del intervalo a evaluar (x1): ')
        try:

```

```

        float(x0)
        float(x1)
    except (ValueError,NameError):
        print('\n Debe introducir valores numéricos! \n')
        main()
    graf_funcion_cuadratica(x0,x1)
elif k == '2':
    x0 = input('Introduzca el límite inferior del intervalo a evaluar (x0): ')
    x1 = input('Introduzca el límite superior del intervalo a evaluar (x1): ')
    try:
        float(x0)
        float(x1)
    except (ValueError,NameError):
        print('\n Debe introducir valores numéricos! \n')
        main()
    graf_funcion_cubica(x0,x1)
elif k == '4':
    t0 = input('Introduzca el límite inferior del intervalo a evaluar (t0): ')
    t1 = input('Introduzca el límite superior del intervalo a evaluar (t1): ')
    print('\n ¿Desea graficar con seno o coseno? ')
    print('"cos" para coseno y "sin"/"sen" para seno \n')
    func = input('Introduzca la opción: ')
    cond = (func == 'sin' or func == 'cos' or func == 'sen' or func == 'Sen' or func == 'Sin' or
func == 'Cos')
    try:
        float(t0)
        float(t1)
        if cond:
            pass
        else:
            print('\n Opcion invalida, volviendo al menu principal \n')
            main()
    except (ValueError,NameError):
        print('\n Debe introducir valores numéricos! \n')
        graf_mov_armonico(t0,t1,func)
elif k == '5':
    print('\n Sintaxis:')
    print('- Al escribir una función como seno debe escribir p.sin (de la forma
p.nombre_de_función)')
    print('- No debe poner signos de igual ("=") en el input')
    print('- Los exponentes deben escribirse con "***" en vez de "^" ')
    print('- Si dos elementos se multiplican, debe de poner signo de multiplicación entre ellos
"***")
    print('\t ejemplo: en vez de 3x debe escribir 3*x')
    print('- Debe usar la variable "x", otras variables no están permitidas')

```

```

print('- La función está automáticamente igualada a "y" \n')
print('_____')
try:
    x0 = float(input('Introduzca el límite inferior del intervalo a evaluar (x0): '))
    x1 = float(input('Introduzca el límite superior del intervalo a evaluar (x1): '))
    y = str(input('Introduzca la función que desea graficar (despejada para y): '))
except (ValueError,NameError):
    print('\n Debe introducir valores numéricos! \n')
    main()
    graf_funcion_val(x0,x1,y)
elif k == '3':
    print('f(x) = m(x) + b')
    try:
        a = float(input('Introduzca el límite inferior del intervalo a evaluar (x0): '))
        b = float(input('Introduzca el límite superior del intervalo a evaluar (x1): '))
        m = float(input('Introduzca la pendiente de la recta (m): '))
        c = float(input('Introduzca el intersecto en y (b): '))
    except (ValueError,NameError):
        print('\n ¡Debe introducir valores numéricos! \n')
        return
    graf_funcion_lineal(a,b,m,c)
elif k == 'm' or k == 'M':
    main()
elif k == 'c' or k == 'q':
    print('\n Operación cancelada \n')
    return
else:
    print('\n ¡Elija una opción correctamente! \n')
elif k == '2':
    # Menú de ecuaciones
    print('_____')
    print('\n     Resolver ecuaciones')
    print('\n Sintaxis: ')
    print('- Debe introducir la ecuación sin "=" (signos de igual).')
    print('- La ecuación está automáticamente igualada a 0.')
    print('- Si introduce una ecuación no lineal muy compleja, el resultado será complejo y se expresará simbólicamente.')
    print('- Las soluciones se expresarán en una lista.')
    print('- El programa resuelve ecuaciones para múltiples variables, pero sólo resuelve una variable a la vez')
    print('- Al introducir la variable asegúrese de que no haya nada pegado a ella, es decir que sea sólo la variable.')
    print('- Si escribe m en el campo de ecuación sin escribir nada más, el programa volverá al menú principal')
    try:

```

```

eq = str(input('\n Introduzca la ecuación que desea resolver: '))
if eq == 'm' or eq == 'M' or eq == ' m' or eq == ' M':
    main()
v = str(input('Introduzca la variable para la cual desea resolver la ecuación: '))
except
(ValueError,NameError,SyntaxError,TypeError,NotImplementedError,AttributeError):
    print('\n Introduzca los valores correctamente \n')
    return
s = resolver_ecuacion(eq,v)
n = 1
print('\n')
for m in s:
    print('    Solucion {0}: {1} \n'.format(n,m))
    n += 1
    continuar()
elif k == '3':
    # menu de calculo de dígitos
    print('. . .')
    print('\n    Cálculo de dígitos especiales')
    print('\n1 - Raíz de números que no son cuadrados perfectos')
    print('2 - e (base del logaritmo natural)')
    print('3 - π (razón de la circunferencia y el diámetro de un círculo)')
    print('m - Volver al menú principal')
    print('c - Cerrar/cancelar')
    k = input('\n Introduzca la opción que desea: ')
    if k == '1':
        try:
            a = int(input('\n Introduzca el número cuya raíz desea calcular: '))
        except (TypeError,ValueError,NameError):
            print('\n ¡Debe introducir valores numéricos para la raiz que desea evaluar! \n')
            return
        try:
            n = int(input('\n Introduzca la cantidad de dígitos que desea calcular (más de 70 mil
puede tomar mucho tiempo): '))
        except (TypeError,ValueError,NameError):
            print('\n ¡Debe introducir valores enteros para la cantidad de dígitos! \n')
            return
        print('\n Dígitos calculados: {0} de raíz de {1} \n'.format(n,a))
        print(digitos_sqrt_irr(a,n))
        return continuar()
    if k == '2':
        try:
            n = int(input('\n Introduzca cuántos dígitos de e desea calcular: '))
        except (ValueError,NameError):
            print('\n Debe introducir valores enteros \n')

```

```

        return
    print('\n Dígitos calculados: {0} de e \n'.format(n))
    print(digitos_e(n))
    return continuar()
if k == '3':
    try:
        n = int(input('\n Introduzca cuántos dígitos de  $\pi$  desea calcular: '))
    except (ValueError,NameError):
        print('\n Debe introducir valores enteros \n')
        return
    print('\n Dígitos calculados: {0} de  $\pi$  \n'.format(n))
    print(digitos_pi(n))
    return continuar()
if k == 'm' or k == 'M':
    main()
if k == 'c' or k == 'C':
    return
else:
    print('\n Error, volviendo al menú principal \n')
    main()
elif k == '4':
    # Menu de derivación e integración
    print('.....')
    print('\n      Menú de cálculos simbólicos')
    print('1 - Derivar función')
    print('2 - Integrar función')
    print('m - Menú principal')
    print('c - Cancelar/cerrar')
    k = input('\n Introduzca la opción que desea: ')
    if k == '1':
        # Explicación de sintaxis para derivación
        print('\n Sintaxis:')
        print('- Debe introducir el valor como un string (entre comillas)')
        print('- Debe utilizar signos de multiplicacion entre cada elemento que se multiplique por otro')
        print('- NO agregue signos de igual, la funcion está automaticamente igualada a la variable dependiente')
        print('- Las funciones deben introducirse en inglés ("sin" en vez de "sen")')
        try:
            f = str(input('\n Introduzca la función que desea derivar: '))
            var = str(input('\n Introduzca la variable con respecto a la que desea derivar: '))
        except
            (ValueError,NameError,SyntaxError,TypeError,NotImplementedError,AttributeError):
                print('\n Introduzca los valores correctamente, volviendo al menu principal \n')
                main()

```

```

    return
if type(f) != str or type(var) != str:
    print('\n Error, debe introducir un string, volviendo al menu principal')
    main()
print('\n La derivada de {0} con respecto a {1} es: \n'.format(f,var))
print(derivar_funcion(f,var))
print('\n')
return continuar()
if k == '2':
    # Explicación de sintaxis para integración
    print('\n Sintaxis:')
    print('- Debe introducir el valor como un string (entre comillas)')
    print('- Debe utilizar signos de multiplicacion entre cada elemento que se multiplique por
otro')
    print('- NO agregue signos de igual, la funcion está automaticamente igualada a la
variable dependiente')
    print('- Las funciones deben introducirse en inglés ("sin" en vez de "sen")')
try:
    f = str(input('\n Introduzca la funcion que desea integrar: '))
    var = str(input('\n Introduzca la variable con respecto a la que desea integrar: '))
except
(ValueError,NameError,SyntaxError,TypeError,NotImplementedError,AttributeError):
    print('\n Introduzca los valores correctamente, volviendo al menu principal \n')
    main()
    return
if type(f) != str or type(var) != str:
    print('\n Error, debe introducir un string, volviendo al menu principal')
    main()
print('\n La integral de {0} con respecto a {1} es: \n'.format(f,var))
print(integrar_funcion(f,var))
print('\n')
return continuar()
elif k == 'c' or k == 'C':
    print('\n Cancelado \n')
    return
elif k == 'm' or k == 'M':
    print('\n Volviendo al menu principal \n')
    main()
else:
    print('\n Error \n')
    main()
elif k == 'c' or k == 'C':
    return
else:
    print('\n Error, debe elegir una opción válida \n')

```

main()