**Nombre David Caleb Chaparro Orozco Fecha 11/10/2022**

**EXAMEN DE CONOCIMIENTOS EN PYTHON**

1. **¿Python es un lenguaje fuertemente tipado?**

 Si



 No



1. **Para añadir una condición alternativa a una declaración condicional if se utiliza**

 elseif



 elif



 else if



 elsif



1. **¿Python es un lenguaje interpretado?**

 Si



 No



1. **Para mostrar el valor de la posición 2 de un array llamada ‘test’ utilizamos**

 print(test [1])



 print(test[2])



 puts(test[2])



 pp(test[2])



1. **¿Qué tan necesaria la indentación de código en python?**

Es muy necesaria, si no está bien arrojará errores.



 No es necesaria.



 Es opcional, depende del programador.



 El código en python no se puede indentar.



1. **¿Qué es el \_\_init\_\_ y haga un ejemplo de su uso?**

Es un método de POO y es utilizado para crear objetos, esta función es llamada cada vez que se crea un objeto de una clase, porque permite inicializar los atributos de objeto, solo se usan dentro de las clases.

Ejm:

*class* Aritmetica:

    """

    Clase Aritmética para realizar las operaciones de sumar, restar, etc.

    """

*def* \_\_init\_\_(*self*, *operandoA*, *operandoB*):

        self.operandoA = operandoA

        self.operandoB = operandoB

*def* sumar(*self*):

        return *float*(self.operandoA + self.operandoB)

*def* restar(*self*):

        return *float*(self.operandoA - self.operandoB)

*def* multiplicar(*self*):

        return *float*(self.operandoA \* self.operandoB)

*def* dividir(*self*):

        if self.operandoB == 0:

            return 'No se puede dividir por CERO pelotudo'

        return *float*(self.operandoA / self.operandoB)

aritmetica1 = Aritmetica(10,5)

print('Calculadora básica'.center(50,'-'))

print(*f*'Suma: {aritmetica1.sumar()}')

print(*f*'Resta: {aritmetica1.restar()}')

print(*f*'Multiplicación: {aritmetica1.multiplicar()}')

print(*f*'Dividir: {aritmetica1.dividir()*:.3f*}')

1. **¿Cuál de los siguientes es un objeto de tipo diccionario?**

 diccionario = ('Numero': 1, 'Nombre': 'Juan')



 diccionario = {'Numero' -> 1, 'Nombre' -> 'Juan'}



 diccionario = {'Numero': 1, 'Nombre': 'Juan'}



 diccionario = {'Numero' => 1, 'Nombre' => 'Juan'}



1. **¿Que es el Self en python?**

 Es un parámetro opcional de un método.



 Es un tipo de dato.



 Se utiliza para mostrar en consola el valor de una variable o cadena de texto.



 Es una instancia u objeto de una clase.



1. **¿Cuál es la forma correcta de escribir un bucle for?**

 for(a in range[0..3])



 for a in range(0..3)



 for(a=0; a<3; a++)



 for a in range(0, 3)



1. **¿Cuál es la diferencia entre \*args, \*\*kwargs?**

|  |  |
| --- | --- |
| ***\*args*** | ***\*\*kwargs*** |
| Significa pasar un número arbitrario de argumentos | Significa pasar un número arbitrario de argumentos con palabras clave |

1. **¿Qué diferencia hay entre una clase y un objeto?**

 Un objeto es una instancia de una clase.



 Ninguna.



 Una clase es una instancia de un objeto.



 Un objeto no tiene tipo.



1. **El resultado de la declaración print('%.2f' % 1714.666) es**

 1714.66



 1714.67



 1714.0



 1715



1. **Defina una lista e insértele valores:**

lenguajes = ['Python','Java', 'JavaScript', 'SQL']

print(*f*'Sin insertar valores: ', lenguajes)

lenguajes.append('Ruby')

print(*f*'Con nuevos valores: ', lenguajes)

1. **Defina una tupla e insértele valores:**

canciones = ()

canciones = ('Let Me Down Slowly', 'No Friends', 'I Lost A Friend', 'Savior')

print(*f*'Todas las canciones: ', canciones)

nueva\_cancion = ('Jet Black Heart',)

cola\_reproduccion = canciones + nueva\_cancion

print(*f*'Agregando la canción: ', cola\_reproduccion)

print(*type*(canciones))

1. Dado un árbol binario, encuentre el ancestro común más cercano entre dos nodos.

**Inputs:**

70,84,85

* 70,84,78,80
* 70,84,78,76
* 70,49,54,51
* 70,49,37,40
* 70,49,37,22

El árbol correspondiente a estos datos es el siguiente:



**Output:**

ancestor(40,78) = 70

ancestor(51,37) = 49

ancestor(76,85) = 84

Diseñe un API REST que permita:

1. Crear un árbol.

2. Dado un arbol y dos nodos, retorne el ancestro común más cercano.

*class* TreeBinary:

*def* \_\_init\_\_(*self*, *data*):

        self.data = data

        self.left = None

        self.right = None

*def* findPath( *root*, *path*, *k*):

    if root is None:

        return False

    path.append(root.data)

    if root.data == k :

        return True

    if ((root.left != None and findPath(root.left, path, k)) or (root.right!= None and findPath(root.right, path, k))):

        return True

    path.pop()

    return False

*def* ancestro(*root*, *n1*, *n2*):

    path1 = []

    path2 = []

    if (not findPath(root, path1, n1) or not findPath(root, path2, n2)):

        return -1

    i = 0

    while(i < len(path1) and i < len(path2)):

        if path1[i] != path2[i]:

            break

        i += 1

    return path1[i-1]

root = TreeBinary(70)

root.left = TreeBinary(49)

root.left.left = TreeBinary(37)

root.left.left.left = TreeBinary(22)

root.left.left.right = TreeBinary(40)

root.left.right = TreeBinary(54)

root.left.right.left = TreeBinary(51)

root.right = TreeBinary(84)

root.right.right = TreeBinary(85)

root.right.left = TreeBinary(78)

root.right.left.left = TreeBinary(76)

root.right.left.right = TreeBinary(80)

print ("Ancestros de (40, 78) = %d" %(ancestro(root, 40, 78)))

print ("Ancestros de (51, 37) = %d" %(ancestro(root, 51, 37)))

print ("Ancestros de (76, 85) = %d" %(ancestro(root,76,85)))