FACULTAD DE INGENIERÍA
FUNDAMENTOS DE COMPUTACION Y PROGRAMACION

PRUEBA ACUMULATIVA

ASPECTOS GENERALES DE LA PRUEBA

- Lea atentamente la prueba y las instrucciones antes de comenzar a desarrollarla.
- Queda prohibido hablar con los compañeros(as) durante el desarrollo de la PEP.
- La PEP contiene 4 preguntas de desarrollo, con un total de **45** puntos y una exigencia del **60%**
- Tiene un límite de tiempo de **90** minutos para responder.
- Escoja 3 de las 4 preguntas que se le entregan.
- En caso de que responda más de 3 sólo se revisarán las 3 primeras preguntas y se ignorará cualquier desarrollo de la cuarta pregunta.
- El equipo docente tiene la prohibición de responder consultas.
- El/La estudiante que se sorprenda en actos deshonestos será calificado con la nota mínima.
- Los elementos tecnológicos deben permanecer apagados y guardados. Queda absolutamente prohibido el uso todo elemento tecnológico. Su uso puede significar la nota mínima o sanciones mayores.
- El alumno deberá identificarse con su Cédula de Identidad.
- Sobre el escritorio sólo podrá existir lápiz (obligatorio) y goma/lápiz corrector (opcional).
- Complete sus datos personales antes de comenzar la evaluación.
- Considere que la evaluación contempla el código, los comentarios y el seguimiento de las buenas prácticas de programación.
- Responda cada pregunta, a continuación de su enunciado, en el espacio que se le entrega para ello.

NOMBRE	RUT	SECCIÓN	

1. (15 puntos) Se desea construir un programa que dado un entero natural, dibuje un triángulo con caracteres numerales (#) y letras O en un archivo de texto figura.txt, por ejemplo, para entradas de valor 4 y 5 las figuras a escribir en el archivo serían:

Para una entrada de valor 4
0#0# 0#0 0# 0

Para una entrada de valor 5
0#0#0 0#0# 0#0 0# 0

Tras realizar el diagrama de abstracción, se estimó que el proceso sería el siguiente:

- La entrada se realizaría a través de entrada estándar utilizando la función `input()`
- El proceso de generar la figura sería realizado por una función `generarFigura`, que recibirá como entrada el valor entero natural y como salida entregará la figura como un string
- La salida se realizará a través de una función `escribirArchivo`, que recibirá como entrada un string con el nombre del archivo y el string a escribir, y como salida entregará un True una vez que el proceso de escritura haya finalizado con éxito.

Implemente el programa en Python en base a la abstracción entregada.

```
# Función que crea la figura solicitada
# a partir de un entero positivo
# Entrada: Número entero positivo
# Salida: String con la figura solicitada
def crearFigura(numero):
    # Se crea un string vacío para almacenar
    # la figura
    figura = ""
    # Se inicializa el primer iterador
    i = 0
    # Se inicializa el primer while
    # con el objetivo de dibujar cada fila
    while i <= numero :
        # Se inicializa el segundo iterador
        j = 0
        # Se inicializa el segundo while
        # con el objetivo de dibujar cada elemento
        # de la fila, la condición numero - i
        # es para conseguir la forma de triángulo
        while j < numero - i :
            # Para diferenciar las O de las #
            # se crea un par if - else
            # Si el elemento es impar
            if j % 2 == 0 :
                # se dibuja una 'O'
                figura = figura + "O"
            # Si el elemento es par
            else :
                # se dibuja un '#'
                figura = figura + "#"
            # Incremento el iterador j
            j = j + 1
        # Al salir del primer while, se agrega
        # un salto de línea
        figura = figura + "\n"
        # Se incrementa el iterador i
        i = i + 1
    # Se retorna la figura
    return figura

# Función que escribe el contenido de un string
# en un archivo de texto
# Entrada: string a escribir, nombre del archivo (string)
# Salida: Booleano de valor True si el proceso se realiza exitosamente
def escribirArchivo(contenido, nombreArchivo):
    # Se abre el archivo en modo de lectura
    archivo = open(nombreArchivo, "w")
    # Se escribe el contenido en el archivo
    archivo.write(contenido)
    # Se cierra el archivo
```

```
    archivo.close()
    # Se retorna
    return True


# BLOQUE PRINCIPAL

# ENTRADA
# Se solicita el valor de entrada
numero = input("Ingrese un valor entero positivo: ")

# PROCESAMIENTO
# Se construye la figura
figura = crearFigura(numero)

# SALIDA
# Se escribe el resultado en un archivo de texto
exito = escribirArchivo(figura, "figura.txt")
# Si el archivo se ha escrito correctamente
if exito :
    # Se informa al usuario de que el proceso ha sido realizado exitosamente
    print "Se ha escrito el archivo correctamente"
```

2. (15 puntos) Actualmente en Chile, existen dos tipos válidos de patentes:

- Las patentes antiguas, corresponden a patentes con formato de dos letras y cuatro números de formato LLNNNN, y que contemplan como válidas todas las letras del alfabeto. Por ejemplo, una patente válida en el formato antiguo sería: PL3236.
- Las patentes nuevas, correspondientes a patentes con formato de cuatro letras y dos números, de formato LL-LL-NN, y que contemplan como válidas todas las letras del alfabeto exceptuando las vocales. Por ejemplo una patente válida en el formato nuevo sería BXCD23.

Construya un programa en Python que reciba como entrada un archivo “patentes.txt” donde en cada línea exista una patente y reescriba un nuevo archivo “validadas.txt”, dónde se indique para cada patente si esta es de formato antigua, de formato nuevo o inválida, Por ejemplo:

patentes.txt
UK2345
KXJV81
COCA18
VDAQ52
HOLA6141

validadas.txt
UK2345: antigua
KXJV81: nueva
COCA18: inválida
VDAQ52: nueva
HOLA6141: inválida

```
# Función que lee un archivo
# Entrada: nombre del archivo (string)
# Salida: Contenido del archivo (lista de strings)
```

```
def leerArchivo(nombreArchivo):
    # Se abre el archivo en modo de lectura
    archivo = open(nombreArchivo, 'r')
    # Con readlines se obtiene el contenido del archivo
    # como una lista de strings, dónde cada string es una línea del archivo
    contenido = archivo.readlines()
    # Se cierra el archivo
    archivo.close()
    # Se retorna el contenido del archivo
    return contenido


# Función que determina si una patente
# es del formato antiguo
# Entrada: Patente (string)
# Salida: True si la patente es válida, False en caso contrario
def validarAntigua(patente):
    # Si la patente tiene más o menos caracteres
    if len(patente) != 6 :
        # Se entrega un False
        return False
    # Si la patente cumple el formato de dos letras y cuatro números
    elif patente[0:2].isalpha() and patente[2:6].isdigit():
        # Se entrega un True
        return True
    # Si no
    else :
        # Se entrega un False
        return False

# Función que determina si una patente
# es del formato nuevo
# Entrada: Patente (string)
# Salida: True si la patente es válida, False en caso contrario
def validarNueva(patente):
    # Se definen los caracteres válidos
    caracteresValidos = 'BCDFGHJKLMNPQRSTVWXYZ'
    # Si la patente tiene más o menos de seis caracteres
    if len(patente) != 6 :
        # Se entrega un False
        return False
    # Si la patente cumple el formato de cuatro letras y dos números
    elif patente[0:4].isalpha and patente[4:6].isdigit():
        # Se revisa que las letras sean letras válidas
        if patente[0] in caracteresValidos \
           and patente[1] in caracteresValidos\
           and patente[2] in caracteresValidos\
           and patente[3] in caracteresValidos :
            # Si lo son, se retorna True
            return True
```

```
            # En caso contrario
            else:
                # Se retorna False
                return False
        # Si los criterios anteriores no se cumplen
        else :
            # Se retorna un False
            return False

# Función que clasifica listado de patentes
# Entrada: lista de patentes (lista de strings)
# Salida: string de patentes clasificadas
def clasificarPatentes(listaDePatentes):
    # Creo una variable para almacenar la salida
    clasificacion = ""
    # Para cada patente en la lista
    for patente in listaDePatentes :
        # Elimino el salto de línea de la patente
        patente = patente.strip()
        # Si la patente es antigua
        if validarAntigua(patente):
            # se clasifica como antigua
            clasificacion = clasificacion + patente + ": antigua\n"
        # Si la patente es nueva
        elif validarNueva(patente):
            # Se clasifica como nueva
            clasificacion = clasificacion + patente + ": nueva\n"
        # Si no es antigua, ni nueva
        else :
            # Se clasifica como inválida
            clasificacion = clasificacion + patente + ": inválida\n"
    # Se entrega la clasificación completa
    return clasificacion

# Función que escribe el contenido de un string
# en un archivo de texto
# Entrada: string a escribir, nombre del archivo (string)
# Salida: Booleano de valor True si el proceso se realiza exitosamente
def escribirArchivo(contenido, nombreArchivo):
    # Se abre el archivo en modo de lectura
    archivo = open(nombreArchivo, "w")
    # Se escribe el contenido en el archivo
    archivo.write(contenido)
    # Se cierra el archivo
    archivo.close()
    # Se retorna
    return True
```

```
# BLOQUE PRINCIPAL

# ENTRADA
patentes = leerArchivo('patentes.txt')

# PROCESO
clasificacion = clasificarPatentes(patentes)

# SALIDA
# Se escribe el resultado en un archivo de texto
exito = escribirArchivo(clasificacion, "validadas.txt")
# Si el archivo se ha escrito correctamente
if exito :
    # Se informa al usuario de que el proceso ha sido realizado exitosamente
       print "Se ha escrito el archivo correctamente"
```

3. **(15 puntos)** Construya un programa en Python que reciba como entrada un string, que solo contiene letras (mayúsculas y minúsculas) y espacios, y muestre por pantalla su traducción a clave Morse, teniendo en cuenta que:

- Para separar en el código una letra de la siguiente se usa /
- Para separar palabras se usa //
- La clave Morse es la siguiente (Sin distinguir mayúsculas de minúsculas):

A •–	J •–––	S •••
B –•••	K –•–	T –
C –•–•	L •–••	U ••–
D –••	M ––	V •••–
E •	N –•	W •––
F ••–•	O –––	X –••–
G ––•	P •––•	Y –•––
H ••••	Q ––•–	Z ––••
I ••	R •–•	

```
# Se define la clave morse como una constante
CLAVE_MORSE = ['.-','-...','-.-.','-..','.','..-.','--.',\
               '....','..','.---','-.-','.-..','--','-.',\
               '---','.--.','--.-','.-.','...','-','..-',\
               '...-','.--','-..-','-.--','--..']
# Se define el alfabeto como una constante
ALFABETO = ['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']
```

```
# Función que traduce un caracter a su equivalente en
# clave morse
# Entrada: un caracter (string)
# Salida: caracter codificado en morse (string)
def traducirCaracter(caracter):
    # Se define un iterador
    i = 0
    # Se itera para revisar todo el alfabeto
    while i < len(ALFABETO) :
        # Si encuentro el caracter en una posición del
        # alfabeto
        if caracter == ALFABETO[i]:
            # Entrego su equivalente en clave morse
            return CLAVE_MORSE[i]
        # Sino, sigo revisando
        i = i + 1

# Función que traduce un mensaje a clave morse
# Entrada: mensaje a traducir (string)
# Salida: mensaje traducido a morse (string)
def traducirMensaje(mensaje):
    # Se crea un string para almacenar el mensaje en Morse
    mensajeMorse = ""
    # Se transforma el mensaje a sólamente mayúsculas
    mensaje = mensaje.upper()
    # Para cada caracter en el mensaje
    for caracter in mensaje :
        # Si el caracter es un espacio
        if caracter == " ":
            # Se añade un / para separar
            mensajeMorse = mensajeMorse + "/"
        # Sino
        else :
            # Se traduce el caracter a morse
            codificado = traducirCaracter(caracter)
            # Se añade el caracter codificado al mensaje
            mensajeMorse = mensajeMorse + codificado + "/"
    # Se entrega el mensaje codificado
    return mensajeMorse

# BLOQUE PRINCIPAL

# ENTRADA
# Se solicita el mensaje a ocultar
mensajeAOcultar = raw_input("Ingrese el mensaje a ocultar: ")

# PROCESAMIENTO
# Se traduce el mensaje
mensajeOculto = traducirMensaje(mensajeAOcultar)
```

```
# SALIDA
# Se entrega el mensaje codificado
print mensajeOculto
```

4. (15 puntos) Para aproximar el número Pi podemos utilizar la serie infinita denominada "serie de Leibniz" que converge a $\pi/4$ para $n = \infty$:

$$\sum_{i=0}^{n} \frac{(-1)^i}{2i+1} = \frac{\pi}{4}$$

Si n es un número natural pequeño la aproximación no es tan buena, por lo que se requiere utilizar soluciones computacionales para aproximar a un valor de Pi adecuado. A fin de determinar cuándo las aproximaciones de la serie de Leibniz son correctas, construya un programa en Python que aproxime el valor de Pi considerando una cantidad dada de n términos.

Junto con el valor de la aproximación, el programa además debe indicar el error absoluto y el error relativo porcentual de la solución aproximada en función del valor de Pi que entrega el módulo math.

Recuerde que las fórmulas para el cálculo de error son:

Error absoluto:

$$error\ Absoluto = |Valor\ Aproximado - Valor\ Real|$$

Error relativo porcentual:

$$errorRelativo = \frac{error\ Absoluto}{Valor\ Real} * 100$$

```
# Desde el módulo math se importa pi
from math import pi

# Función que calcula la serie de Leibniz con n términos
# Entrada: Cantidad de iteraciones de la serie (entero)
# Salida: Valor de la serie aproximada (float)
def calculaSerieLeibniz(iteraciones):
    # Variable para almacenar el resultado
    resultado = 0
    # Iterador para realizar las iteraciones requeridas
    # Se inicia en 0.0 para asegurar que todos los cálculos parciales
    # sean float
    i = 0.0
    # Mientras no se alcance el número de iteraciones requeridas
    while i <= iteraciones :
        # Se calcula el valor del término
```

```
        termino = (-1)**i /(2 * i + 1)
        # Se añade el término calculado al resultado
        resultado = resultado + termino
        # Se incrementa el iterador
        i = i + 1
    # Se retorna el resultado
    return resultado

# Función que calcula el error absoluto
# Entrada: Dos valores numéricos (valor aproximado y real)
# Salida: Valor del error absoluto
def calculaErrorAbsoluto(valorAproximado, valorReal):
    # Se calcula el error absoluto restando ambos valores
    errorAbsoluto = valorAproximado - valorReal
    # Para entregar un valor absoluto se revisa si el cálculo es menor a cero
    if errorAbsoluto < 0 :
        # En caso de serlo, se entrega con el signo cambiado
        return - errorAbsoluto

    # De lo contrario se entrega como fue calculado
    else :
        return errorAbsoluto

# Función que calcula el error relativo porcentual
# Entrada: Dos valores numéricos (valor real y error absoluto)
# Salida: Valor del error relativo porcentual
def calculaErrorRelativo (valorReal, errorAbsoluto):
    # Se realiza el cálculo de la fórmula de error relativo porcentual
    # se multiplica por 100.0 para asegurar que el resultado sea flotante
    errorRelativo = errorAbsoluto * 100.0 / valorReal
    # Se retorna el error relativo
    return errorRelativo

# BLOQUE PRINCIPAL

# ENTRADA
# Se solicita el número de iteraciones a evaluar
iteraciones = input("Ingrese la cantidad de iteraciones: ")

# PROCESAMIENTO

# Se calcula el resultado de la serie de Leibniz
leibniz = calculaSerieLeibniz(iteraciones)
# Se calcula el valor de PI a partir del resultado anterior
piAproximado = leibniz * 4
# Se calcula el error absoluto
errorAbsoluto = calculaErrorAbsoluto(piAproximado, pi)
# Se calcula el error relativo porcentual
errorRelativo = calculaErrorRelativo(pi,errorAbsoluto)
```

```
# SALIDA

# Se entregan los cálculos realizados
print "El valor de PI, con :",iteraciones,"iteraciones, es:"
print piAproximado
print "con un error absoluto de :", errorAbsoluto, "y "
print "error relativo de: ", errorRelativo, "%"
```