Universidad del Desarrollo
Facultad de Ingeniería

Taller de Programación

Examen

2 de julio de 2019

Instrucciones:

- El certamen contiene 3 problemas y un bonus. Lea atentamente el enunciado de cada uno de ellos.
- Para cada problema cree un archivo.py distinto. El nombre del archivo debe ser el número del problema. Por ejemplo: uno.py, dos.py, tres.py o bonus.py
- Suba sus respuestas como un archivo **ZIP** a la sección Evaluación en http://canvas.udd.cl. Solo tendrá una oportunidad para subir sus respuestas.
- Recuerde que usaremos un software de detección de plagio, confiamos en su honestidad.
- Tiempo total: **2 horas y 50 minutos**.

(2pts) 1. Torres de celular

Movistar la acaba de contratar como ingeniera de software para mejorar la cobertura de la señal en lugares apartados. Su tarea es implementar la clase `Torre` que permite saber si una localidad está o no bajo la cobertura de una torre. Para esto debe seguir la siguiente API:

API Torre:

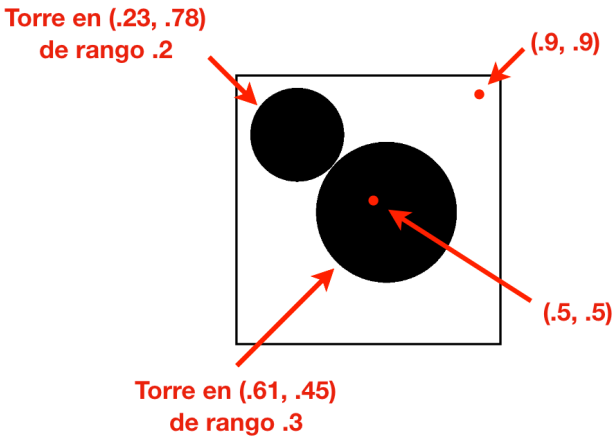
class	Torre	
	Torre(x, y, r)	Crea una torre en (x, y) con un rango de cobertura r (0.2pts)
bool	enRango(x0, y0)	Retorna True si el punto (x0, y0) está en la cobertura de esta torre, y Falso en otro caso (0.8pts)

Asuma que el rango de cobertura es un círculo de radio `r` con centrado en la posición de la torre. Luego, la implementación de `enRango` es chequear si la distancia entre el punto y el centro de la torre es menor al radio `r`. Específicamente, utilice la siguiente ecuación:
$\sqrt{(x_0 - x)^2 + (y_0 - y)^2} < r$.

Para este ejercicio debe:

1. (1pto) Implementar la clase `Torre`, usando la ecuación entregada para `enRango`
2. (1pto) Crear una código para probar su implementación siguiendo este procedimiento:
    - Crear una lista de dos torres: una en (.61, .45) de rango .3 y otra en (.23, .78) de rango .2
    - Chequear que el punto (.5, .5) está en el rango de cobertura de alguna torre
    - Chequear que el punto (.9, .9) NO está en el rango de cobertura de alguna torre
    - Su código debería imprimir una línea por cada punto, indicando si está o no en la cobertura de una torre. Por ejemplo:

```
(0.5, 0.5) tiene cobertura
(0.9, 0.9) NO tiene cobertura
```



Solución posible:

```python
from math import sqrt

class Torre:
    def __init__(self, x, y, r):
        self.x = x
        self.y = y
        self.r = r
    def enRango(self, x0, y0):
        return sqrt((x0-self.x)**2 + (y0-self.y)**2) < self.r


t1 = Torre(.61, .45, .3)
t2 = Torre(.23, .78, .2)
ts = [t1, t2]

p1 = (.5, .5)
p2 = (.9, .9)
ps = [p1, p2]
for p in ps:
    r = False
    for t in ts:
        if t.enRango(p[0],p[1]):
            r = True
    if r:
        print(p, 'en rango')
    else:
        print(p, 'no en rango')

```

- 0.2pts por hacer el constructor en Torre
- 0.8pts por hacer la función enRango
- 1pts si hace un ciclo o un if con un or para chequear si un punto está en alguna torre.

- -0.5 pts si imprime `en rango` dos veces (una para cada torre)

(2pts) 2. Contraseña segura

Cree una función que chequee si un string es una "**buena**" contraseña. Una "**buena**" contraseña se define como:

1. (0.5pts) Tiene al menos 8 caracteres de largo
2. (0.5pts) No es un string dentro de una lista de palabras
3. (0.5pts) No es un string dentro de una lista de palabras seguida por un número entero (ejemplo: hola29226)
4. (0.5pts) No es la concatenación de dos strings en una lista de palabras (ejemplo: holaqwerty)

La función debe retornar True si el string de entrada es **buena** contraseña, y False en caso contrario.

Asuma que la lista de palabras prohibidas es la siguiente:

```
palabras =
['de','la','que','el','en','como','hola','password','progradud','qwerty','abc',
'mimamamemima']
```

Hint: utilice el método `is_digit()` para saber si un string es un número, se utiliza de la siguiente manera:

```
s = '123'
if s.is_digit() == True:
  print('el string s es un número entero')
```

Solución posible:

```
palabras =
['de','la','que','el','en','como','hola','password','progradud','qwerty','abc',
'mimamamemima']

def buena(s):
    if len(s) < 8:
        return False

    if s in palabras:
        return False

    n = len(s)
    if s[:n] in palabras:
        if s[n+1:].is_digit():
            return False

    pp = []
```

```
    for a in palabras:
        for b in palabras:
            pp.append(a+b)

    if s in pp:
        return False

    return True

if buena('miau') == False:
    print('correcto')
if buena('prograudd') == False:
    print('correcto')
if buena('hola29226') == False:
    print('correcto')
if buena('holaqwerty') == False:
    print('correcto')
if buena('Qwa#$%a2') == True:
    print('correcto')
```

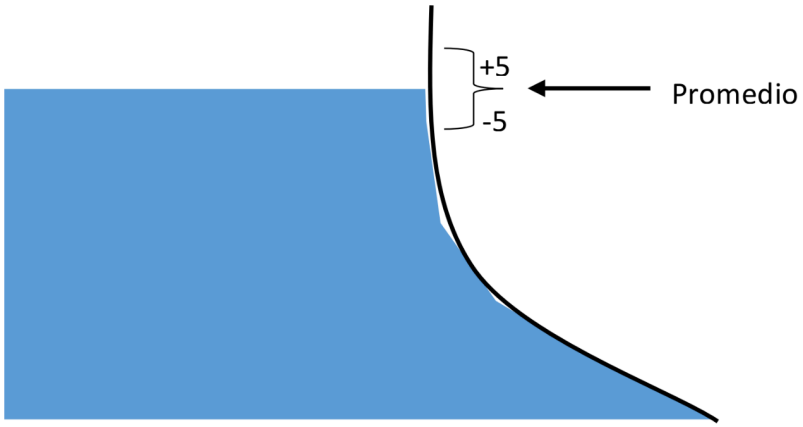
- 0.5 por cada condición correctamente implementada

(2pts) 3. Nivel del agua

Una métrica importante para asegurar la calidad en la generación de energía hidroeléctrica es mantener un nivel estable de agua dentro de una represa. Para medir la calidad de generación durante un año, el nivel del agua se debe mantener más o menos 5 metros con respecto al promedio del año.

Para este programa usted debe hacer la función `calidad()` que realice lo siguiente:

1. (0.5pts) Lea el nivel de altura del agua los 365 días del año y almacenarlos en una lista. Cada nivel está en metros sin decimales.
2. (0.5pts) Determinar e imprimir el valor promedio del nivel del agua en los 365 días.
3. (1.0pto) Imprimir los días del año en el cual el nivel estaba **fuera del rango** +-5 respecto al promedio. El día puede ser impreso como valor numérico, es decir el 11 de Febrero es el número 42 y el 1 de Enero es el número 1.



Solución posible:

```python
def calidad():
    nivel = []
    for i in range(365):
        x = int(input())
        nivel.append(x)
    prom = sum(nivel)/len(nivel)

    for i in range(365):
        if nivel[i] > prom+5:
            print(i+1)
        else if nivel[i] < prom-5:
            print(i+1)
```

- 0.5 pregunta por los valores de cada día
- 0.5 si calcula el promedio
- 0.7 si compara valores +-5 del promedio

- 0.3 si imprime el í-ésimo día (1 de enero es el día 1)

(1 pto) Bonus: sub-strings de tamaño k

Cree la función `substrings` que reciba como entrada un texto s y un entero k, y que imprima los sub-strings únicos de tamaño k que contiene el texto s, uno por linea. Por ejemplo, si el texto de entrada es `s='ABABBBABAB'` y `k=3`, existen 5 sub-strings de tamaño 3: `ABA BAB BBB ABB BBA`. Este cálculo es útil para comprimir pues permite reemplazar los substrings de tamaño `k` por una palabra más pequeña.

La función substrings debe imprimir un substring por línea, recuerde que deben ser distintos.

Hint: use el operador corchete para `s[i:i+k]` para obtener el i-ésimo substring de tamaño `k`.

Solución posible:

```
def kreps(s, k):
    w = set()
    for i in range(len(s)-k):
        w.add(s[i:i+k])
    for p in w:
        print(p)

kreps('ABABBBABAB', 3)
```

- 0.5pts si recorre correctamente el string
- 0.5pts si obtiene los substrings unicos: puede usar un set, un dict o una lista y eliminar duplicados