Python para el análisis de datos- Lectura 4

Ing. Pedro Rotta

Universidad de Piura - Vida Universitaria

Enero-2022

¿Por qué bucles?

Normalmente cuando trabajamos con datos, hacemos actos repititvos.

¿Por qué bucles?

Normalmente cuando trabajamos con datos, hacemos actos repititvos.

Por ejemplo si se entra al *flujo* de un programa, el programa se mantendrá funcionando hasta que el usuario decida cerrarlo.

¿Por qué bucles?

Normalmente cuando trabajamos con datos, hacemos actos repititvos.

Por ejemplo si se entra al *flujo* de un programa, el programa se mantendrá funcionando hasta que el usuario decida cerrarlo.

Por ejemplo, analicemos el primer programa de Colab.

Debugging

La **depuración** es un término que se usa cuando se analiza lo que sucede en el código

Debugging

La **depuración** es un término que se usa cuando se analiza lo que sucede en el código

Por ejemplo en el código anterior, la primera línea identifica lo primero que sucederá, que es guardar en una variable la decision de un usuario(que es una letra o una cadena de texto.)

Debugging

La **depuración** es un término que se usa cuando se analiza lo que sucede en el código

Por ejemplo en el código anterior, la primera línea identifica lo primero que sucederá, que es guardar en una variable la decision de un usuario(que es una letra o una cadena de texto.)

Luego se define el control del flujo. Quiere decir que mientras se cumpla la condición, el flujo del programa será lo que esté dentro del bucle.

Debugging

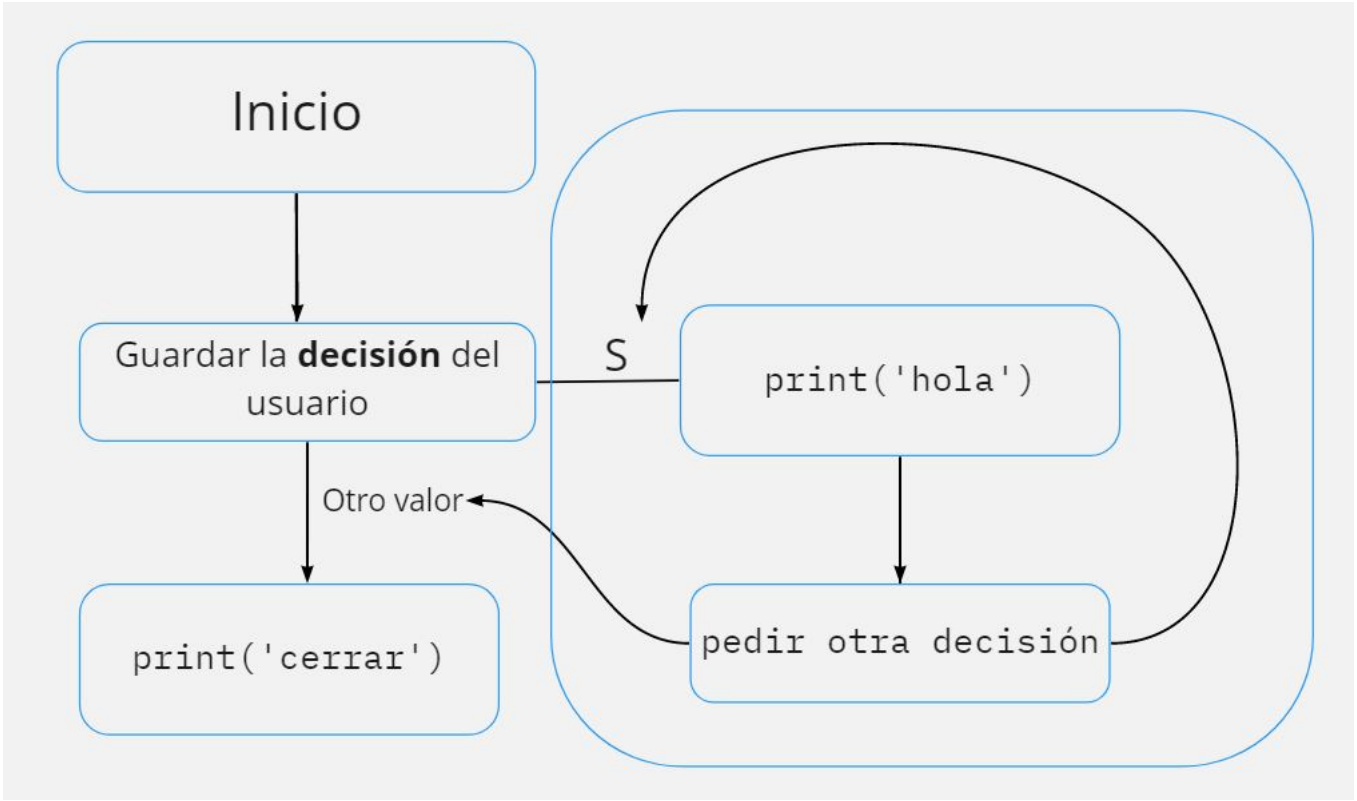
La **depuración** es un término que se usa cuando se analiza lo que sucede en el código

Por ejemplo en el código anterior, la primera línea identifica lo primero que sucederá, que es guardar en una variable la decision de un usuario(que es una letra o una cadena de texto.)

Luego se define el control del flujo. Quiere decir que mientras se cumpla la condición, el flujo del programa será lo que esté dentro del bucle.

Cuando la condición cambia, el bucle se termina, por lo que se sigue el flujo normal.

En esta imagen podemos ver un esquema de lo que está pasando.



While

El primer comando de control es **while**, cuyo significado sería *mientras*. Su sintaxis es:

```
initial condition
while statement:
  command 1
  command 2
  ...
  condition
```

While

El primer comando de control es **while**, cuyo significado sería *mientras*. Su sintaxis es:

```
initial condition
while statement:
    command 1
    command 2
    ...
    condition
```

Que quiere decir que mientras se cumpla la condición se cumplirán los comandos dentro.

While

El primer comando de control es **while**, cuyo significado sería *mientras*. Su sintaxis es:

```
initial condition
while statement:
  command 1
  command 2
  ...
  condition
```

Que quiere decir que mientras se cumpla la condición se cumplirán los comandos dentro.

Cuidado: Debes tener en cuenta que al final del último command, tienes que volver a verificar la condición, sino puedes tener un bucle infinito.

Casos particulares

El uso del While funciona bien para estos algoritmos particulares.

- Cambios de flujo del programa.Por ejemplo Salir de un programa. (Problema 1 de Colab)

Casos particulares

El uso del While funciona bien para estos algoritmos particulares.

- Cambios de flujo del programa.Por ejemplo Salir de un programa. (Problema 1 de Colab)
- Cuando no se sabe de antemano cuántos elementos se tendrán. (Problema 2 de Colab)

Casos particulares

El uso del While funciona bien para estos algoritmos particulares.

- Cambios de flujo del programa.Por ejemplo Salir de un programa. (Problema 1 de Colab)
- Cuando no se sabe de antemano cuántos elementos se tendrán. (Problema 2 de Colab)
- Cuando el flujo está determinado por un límite máximo o mínimo (Problema 3 de Colab)

For

El segundo controlador de flujo es **for** que significa para y su sintaxis es

:

```
collection
for i in collection:
  command 1
  command 2
  ...
```

For

El segundo controlador de flujo es **for** que significa para y su sintaxis es

:

```
collection
for i in collection:
  command 1
  command 2
  ...
```

Que quiere decir que para todos los elemento de la colección se realizarán los comandos. (Ver el problema 4 de Colab)

Comandos para loops

Algunos comandos importantes para loops son:

- **range**: Es para definir un conjunto finito de elementos desde un valor inicial a uno final, por ejemplo:

```
for i in range(0,100) :
...
```

(Ver problema 5 de colab)

Comandos para loops

Algunos comandos importantes para loops son:

- **range**: Es para definir un conjunto finito de elementos desde un valor inicial a uno final, por ejemplo:

```
for i in range(0,100) :
...
```

  (Ver problema 5 de colab)

- **len**: Es para definir la longitud de una colección, por ejemplo:

```
for i in range(0,len(colection) :
...
```

  (Ver Problema 6 de colab)

Comandos de interrupción

Un comando de interrupción corta el flujo normal del loop y lo redirecciona. Los más importantes son:

- **Break**: Corta por completo el loop, sale del loop y sigue el flujo del programa luego del loop.(Ver problema 7)

Comandos de interrupción

Un comando de interrupción corta el flujo normal del loop y lo redirecciona. Los más importantes son:

- **Break**: Corta por completo el loop, sale del loop y sigue el flujo del programa luego del loop.(Ver problema 7)
- **Continue**: Sigue a la siguiente iteración del loop sin tomar en cuenta el proceso de esa iteración. (Ver problema 8)
- **Pass**: No hace nada con esta iteración, pero evita un vacío que cause errores. (Ver problema 8)

Casos particulares

For se usa para los siguientes casos particulares:

- Para algoritmos de búsqueda simples. (Ver problema 9)

Casos particulares

For se usa para los siguientes casos particulares:

- Para algoritmos de búsqueda simples. (Ver problema 9)
- Para procesos que requieran analizar todo un rango de elementos y que no tengan límites en el flujo. (Ver problema 10)

Casos particulares

For se usa para los siguientes casos particulares:

- Para algoritmos de búsqueda simples. (Ver problema 9)
- Para procesos que requieran analizar todo un rango de elementos y que no tengan límites en el flujo. (Ver problema 10)
- Para analizar colecciones de elementos. (Ver problema 11)

Funciones

Las funciones son la tercera forma de controlar un flujo.

Se usan para hacer el flujo más sencillo, reducen el código y además son rehusables, por lo que son comunmente usadas. Realmente todos los programas se escriben como funciones.

Funciones

Las funciones son la tercera forma de controlar un flujo.

Se usan para hacer el flujo más sencillo, reducen el código y además son rehusables, por lo que son comunmente usadas. Realmente todos los programas se escriben como funciones.

Sintaxis:

```
def name(atributes):
    code here
```

Reglas para funciones

- Toda función debe tener cómo máximo una extensión de 15 lineas para que esté bien escrita
- La función main() es la función principal del código y responde al flujo principal.
- Deben estar debidamente comentadas
- Su nombre comienza en minúscula, el caso de ser compuesto, cada letra inicial es mayúscula

Ver problema 12 en Colab.

Alcance de una variable

El **Scope** o en español *alcance* de una variable.Se define como la importancia de la variable en el flujo del programa. Normalmente hay dos tipos de alcance.

- **Variables Globales**: Son variables que tienen importancia en cualquier parte del flujo, es decir se pueden definir fuera y dentro de cualquier función.
- **Variables locales**: Son variables que tienen importancia solo en una función en específico, Si se llaman en otra parte del flujo, no se podrán llamar.

Método Return: el método `return` devuelve el valor de una variable definida en una función. Es la respuesta final de la función y siempre se debe usar porque las variables definidas dentro de la función tienen alcance local.

Ejemplos

En los siguientes ejemplos de Colab, podemos practicar lo aprendido en clase