IWI 131-PROGRAMACIÓN

Ayudantía 12

Ayudante: Anastasiia Fedorova

Paralelo: 212

Fecha: 22.07.2020

TRABAJO CON STRINGS

Ya hemos visto algunos de los operadores, que podemos aplicar sobre los strings – por ejemplo, la concatenación, la repetición, tomar un sub-string (slice), in, etc.

Sin embargo, existen muchas más herramientas para trabajo con strings. Python, de hecho, es muy usado en text analytics (text mining), automatización y data science.

```
[a] – acceso por índice
[a:b:c] – slice en rango con cierto paso
in – True si elemento está en string, en otro caso False
not in – True si elemento no está en string, en otro caso False
+ – concatenación
* – repetición (concatenación consigo mismo)
```

REPLACE()

El método replace() retorna **una copia** de string viejo, pero con substring sustituido.

```
original = "me gusta programar"
nuevo = original.replace(" ", "_")
print(nuevo)
>>> me_gusta_programar
```

.UPPER() Y .LOWER()

En Python, podemos cambiar el texto de modo que tenga todas las letras mayúsculas o todas minúsculas, utilizando el método `.upper()` o `.lower()`, respectivamente.

```
string = "Programacion"
print(string.upper())
>>> PROGRAMACION
```

```
string = "Programacion"
print(string.lower())
>>> programacion
```

.JOIN()

`s.join(lista_de_strings)` permite unir la lista de string, usando el s como “pegamento”.

```
lista1 = ["iwi", "1", "3", "1"]
print("".join(lista1))
>>> iwi131

lista2 =["1", "2", "3"]
print("-->".join(lista2))
>>> 1-->2-->3
```

.SPLIT()

`s.split()` separa el strings en varios strings, usando los espacios en blanco como separador. El valor retornado es **una lista** de strings.

```
oracion1 = "Me gusta Python, pero prefiero C++"
print(oracion1.split())
>>> ["Me", "gusta", "Python,", "pero", "prefiero", "C++"]
```

Además, es posible pasar un parámetro al método `split` que indica cuál será el separador a usar (en vez de los espacios en blanco)

```
print(oracion1.split(","))
>>> ["Me gusta Python", "pero prefiero C++"]
```

LIST()

Al aplicar `list()` sobre un string, obtenemos el conjunto de las letras y todos los simbolos especiales, que forman parte del string.

```
print(list("Python es facil"))
>> > ["P", "y", "t", "h", "o", "n", " ",
     "e", "s", " ",
     "f", "a", "c", "i", "l"]
```

SALTOS DE LÍNEA

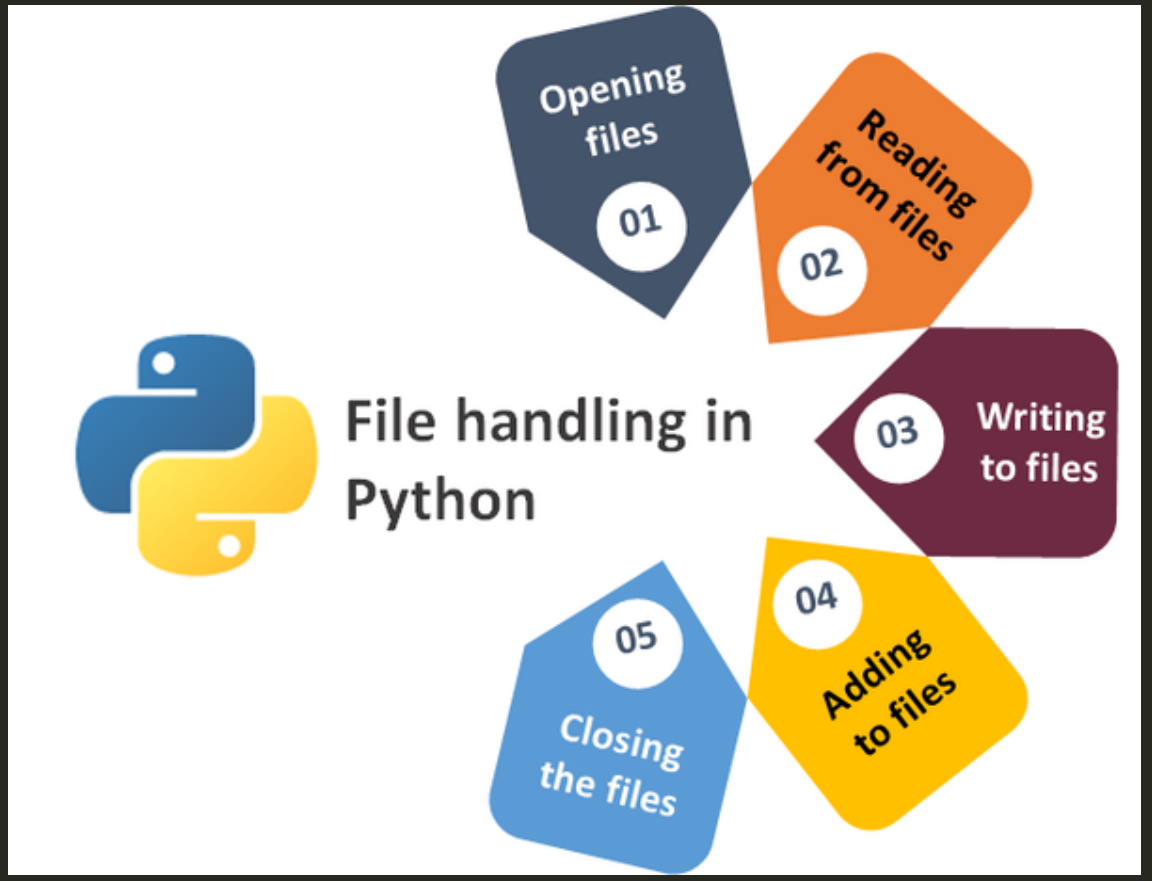
Un string puede contener caracteres de **salto de línea**, que tienen el efecto equivalente al de presionar la tecla Enter. El carácter de salto de línea se representa con \n. Los saltos de linea **se cuentan** en el largo de string, como 1 solo carácter.

```
a = "hurones\ngatitos\nperritos"
>>> hurones
>>> gatitos
>>> perritos
```

ARCHIVOS

Ahora que vimos que strings tienen muchas operaciones útiles, tenemos que preguntarnos: ¿habrá alguna manera de procesar el texto de la fuente externa (no un input de consola)?

En efecto, Python provee varias heramientas de lectura de archivos.



5 pasos en lectura de archivos usando Python.

LECTURA DE ARCHIVOS

Para leer datos de un archivo, hay que abrirlo de la siguiente manera:

```
archivo = open(nombre)
```

nombre es un string que tiene el **nombre del archivo**. archivo es el archivo lógico a través del que se manipulará el archivo real.

Si el archivo no existe, ocurrirá un **error de entrada y salida** (**IOError**).

Es importante recordar que la variable archivo es una representación abstracta del archivo, y no los contenidos del mismo.

LECTURA DE ARCHIVOS

La manera más simple de leer el contenido es hacerlo línea por línea. Para esto, basta con poner el archivo lógico en un ciclo for:

```
for linea in archivo:
    # hacer algo
```

Una vez que los datos han sido leídos del archivo, hay que cerrarlo:

```
archivo.close()
```

LECTURA DE ARCHIVOS

Por ejemplo, supongamos que tenemos el
archivo ramos.txt que tiene el siguiente contenido:

```
iwi131
mat21
efi100
hrw131
```

El siguiente programa imprime cuántos símbolos hay en cada línea:

```
archivo = open('ramos.txt')
for linea in archivo:
    print len(linea)
archivo.close()
```

```
>>> 7
>>> 7
>>> 7
>>> 7
```

Note que el salto de línea (el "enter") es considerado en la cuenta.

```
+---+---+---+---+---+---+---+
| i | w | i | 1 | 3 | 1 |\n | = 7 simbolos
+---+---+---+---+---+---+---+
```

.STRIP()

Para obtener el string sin el salto de línea, se puede usar el método strip, que elimina todos los símbolos de espaciado al principio y al final del string:

```
s = '    iwi131\n'
print(s.strip())
>>>'iwi131'
```

MODOS DE LECTURA

Los ejemplos anteriores suponen que el archivo por leer existe, y está listo para ser abierto y leído. Sin embargo, es posible especificar que es necesario crear un archivo nuevo para re-escribir el anterior, que se desea solo leer los contentidos o inlcluso que deseamos agregar nuevo contenido al final de nuestro archivo original.

MODOS DE LECTURA

Existen más modos de lectura, pero en el curso nos importan los siguientes 3:

Modo	Descripción
r	**Abre un archivo solo para lectura**
w	**Abre un archivo solo para escritura**. Sobreescribe el archivo si este ya existe. Si el archivo no existe, crea un nuevo archivo para escritura.
a	**Abre un archivo para anexo**. El puntero del archivo esta al final del archivo si este existe. Es decir, el archivo está en modo anexo. Si el archivo no existe, crea un nuevo archivo para escritura.

ESCRITURA DE ARCHIVOS

Para escribir en un archivo, este debe ser abierto en modo “a” o “w”. Una vez abierto el archivo, uno puede escribir datos en él usando el método `write`:

```
a = open('prueba.txt', 'w')
a.write('Hola ')
a.write('mundo.')
a.close()
```

```
Hola mundo.
```

EJERCICIO 1

El archivo datos1.txt tiene tres números enteros en cada línea:

```
45 12 98
1 12 65
7 15 76
54 23 1
65 2 84
```

A)Escriba la función suma_lineas(nombre_archivo) que entregue una lista con las sumas de todas las líneas del archivo

```
suma_lineas('datos1.txt')
>>>[155, 78, 98, 78, 151]
```

SOLUCIÓN

```
def suma_lineas(archivo):
    a = open(archivo, "r")
    lista = []
    for linea in a:
        valores = linea.strip().split(" ")
        aux_sum = 0
        for i in valores:
            aux_sum += int(i)
        lista.append(aux_sum)
    a.close()
    return lista
```

EJERCICIO 1

El archivo datos1.txt tiene tres números enteros en cada línea:

```
45 12 98
1 12 65
7 15 76
54 23 1
65 2 84
```

B)Escriba la función suma_columnas(nombre_archivo) que entregue una lista con las sumas de las tres columnas del archivo:

```
suma_columnas('datos1.txt')
>>>[172, 64, 324]
```

SOLUCIÓN

```
def suma_columnas(archivo):
    a = open(archivo, "r")
    # Nota: archivo de 3 columnas.
    # La suma-por-columna mas general tiene otra
implementacion.
    lista = [0, 0, 0]
    for linea in a:
        valores = linea.strip().split(" ")
        for i in range(0, 3):
            lista[i] += int(valores[i])
    a.close()
    return lista
```

EJERCICIO 2

Una tienda tiene la información de sus productos en un archivo llamado productos.txt. Cada línea del archivo tiene tres datos: el código del producto (un número entero), el nombre del producto, y la cantidad de unidades del producto que quedan en bodega.

Los datos están separados por un símbolo /. Por ejemplo, el siguiente puede ser el contenido del archivo:

```
1265/Reloj/26
613/Cuaderno/87
9801/Vuvuzela/3
321/Lápiz/12
5413/Tomate/5
```

EJERCICIO 2

Escriba la función existe_producto(codigo) que indique si existe el producto con el código dado:

```
>>> existe_producto(1784)
False
>>> existe_producto(321)
True
>>> existe_producto(613)
True
>>> existe_producto(0)
False
```

SOLUCIÓN

```
def existe_producto(codigo):
    a = open("productos.txt", "r")
    for linea in a:
        cod, _, _ = linea.strip().split("/")
        if int(cod) == codigo:
            return True
    a.close()
    return False
```

EJERCICIO 2

Escriba la función por_reabastecer() que cree un nuevo archivo llamado por_reabastecer.txt que contenga los datos de todos los productos de los que queden menos de 10 unidades.

En este caso, el archivo por_reabastecer.txt debe quedar así:

```
9801/Vuvuzela/3
5413/Tomate/5
```

SOLUCIÓN

```
def por_reabstecer():
    a = open("productos.txt", "r")
    reabastecer = open("por_reabstecer.txt", "w")
    for linea in a:
        cod, nombre, unidad = linea.strip().split("/")
        if int(unidad) < 10:
            reabastecer.write(linea)
    a.close()
    reabastecer.close()
```

progra.usm.cl

EJERCICIO 3

Una consulta médica tiene un archivo pacientes.txt con los datos personales de sus pacientes. Cada línea del archivo tiene el rut, el nombre y la edad de un paciente, separados por un símbolo :. Así se ve el archivo:

```
12393241-2:Ignacio Rubio:33
11426761-9:Romina Pérez:35
15690109-1:Francisco Ruiz:26
6092377-9:Alfonso San Martín:65
9023365-3:Manuel Toledo:38
...
```

EJERCICIO 3

Además, cada vez que alguien se atiende en la consulta, la visita es registrada en el archivo atenciones.txt, agregando una línea que tiene el rut del paciente, la fecha de la visita (en formato dia-mes-año) y el precio de la atención, también separados por :. El archivo se ve así:

..

```
8015253-1:4-5-2010:69580
12393241-2:6-5-2010:57274
10985778-5:8-5-2010:73206
8015253-1:10-5-2010:30796
8015253-1:12-5-2010:47048
12028339-1:12-5-2010:47927
...
```

EJERCICIO 3

A) Escriba la función costo_total_paciente(rut) que entregue el costo total de las atenciones del paciente con el rut dado:

```
>>> costo_total_paciente('8015253-1')
297572
>>> costo_total_paciente('14350739-4')
0
```

B) Escriba la función pacientes_dia(dia, mes, anio) que entregue una lista con los nombres de los pacientes que se atendieron el día señalado

```
>>> pacientes_dia(2, 6, 2010)
['Pablo Muñoz', 'Alfonso San Martín']
>>> pacientes_dia(23, 6, 2010)
[]
```

SOLUCIÓN

```
def costo_total_paciente(rut):
    a = open("atenciones.txt", "r")
    suma = 0
    for linea in a:
        r, _, costo = linea.strip().split(":")
        if rut == r:
            suma += int(costo)
    a.close()
    return suma
```

SOLUCIÓN

```
def pacientes_dia(dia, mes, anio):
    a1 = open("atenciones.txt", "r")
    lista_ruts = []
    lista_pacientes = []
    for linea in a1:
        rut, fecha, _ = linea.strip().split(":")
        d, m, a = fecha.split("-")
        if int(d) == dia and int(m) == mes and int(a) == anio:
            lista_ruts.append(rut)
    a1.close()
    a2 = open("pacientes.txt", "r")
    for linea in a2:
        rut, nombre, _ = linea.strip().split(":")
        if rut in lista_ruts:
            lista_pacientes.append(nombre)
    a2.close()
    return lista_pacientes
```

EJERCICIO 3

C) Escriba la función separar_pacientes() que construya dos nuevos archivos: jovenes.txt, con los datos de los pacientes menores de 30 años; mayores.txt, con los datos de todos los pacientes mayores de 60 años. Ej. `jovenes.txt`:

```
13087677-3:Jorge Álvarez:28
12028339-1:Jorge Argandoña:29
...
```

D) Escribir una función ganancias_por_mes() que construya un nuevo archivo llamado ganancias.txt que tenga el total de ganancias por cada mes en el siguiente formato:

```
5:933159
6:1120967
7:124903
```

SOLUCIÓN

```
def separar_pacientes():
    a = open("pacientes.txt", "r")
    jovenes = open("jovenes.txt", "w")
    mayores = open("mayores.txt", "w")
    for linea in a:
        rut, nombre, edad = linea.strip().split(":")
        if int(edad) < 30:
            jovenes.write(linea)
        elif int(edad) > 60:
            mayores.write(linea)
    a.close()
    jovenes.close()
    mayores.close()
```

SOLUCIÓN

```
def ganancias_por_mes():
    a = open("atenciones.txt", "r")
    por_mes = []
    for i in range(0, 12):
        por_mes.append(0)
    for linea in a:
        _, fecha, costo = linea.strip().split(":")
        fecha = fecha.split("-")
        mes = int(fecha[1])
        por_mes[mes - 1] += int(costo)
    a.close()
    ganancia = open("ganancias.txt", "w")
    for i in range(0, 12):
        if por_mes[i] > 0:
            nueva_linea = ":".join([str(i + 1), str(por_mes[i])])
            ganancia.write(nueva_linea + "\n")
    ganancia.close()
```