Clase 4: Un poco más de Python

Grupo de Desarrollo Lihuen

Año 2012

Tipos de Datos

Defínción

Definición

Un Tipo de Datos define el rango de valores que puede tomar una variable y el conjunto de operaciones que pueden aplicarse sobre la misma.

Algunos lenguajes, como Python, realizan **conversiones implícitas** de tipos que debemos tener presentes a la hora de desarrollar nuestros algoritmos.

- x=7/2
- x=7/2.0

¿Qué pasa en estas dos asignaciones?

De acuerdo al tipo de datos de los operandos, Python resuelve la operación de una u otra forma.

Tipos de Datos

Defínción

Definición

Un Tipo de Datos define el rango de valores que puede tomar una variable y el conjunto de operaciones que pueden aplicarse sobre la misma.

Algunos lenguajes, como Python, realizan **conversiones implícitas** de tipos que debemos tener presentes a la hora de desarrollar nuestros algoritmos.

- x=7/2
- x=7/2.0

¿Qué pasa en estas dos asignaciones?
De acuerdo al tipo de datos de los operandos, Python resuelve la operación de una u otra forma.

Tipos de Datos

Conversión

Es posible realizar una **conversión explícita** de tipos:

- x=int("3")+1
- x= float(3/5)
- x= 7/int(2.5)

- **int():** Convierte a entero el argumento
- **float():** Convierte a un número flotante
- **str():** Convierte a una cadena de caracteres (string)

Python utiliza un criterio de comparación de cadenas muy natural: el orden alfabético

Observemos ...

- “casa“ es menor que ”fuego“
- ”cielo“ es mayor que ”ciego“

Pero miremos lo siguiente:

- “Casa” es MAYOR que “fuego”
- “cielo” es MENOR que “Ciego”

Las mayúsculas se consideran “más grandes” que las minúsculas...

Python utiliza un criterio de comparación de cadenas muy natural: el orden alfabético

Observemos ...

- “casa“ es menor que ”fuego“
- ”cielo“ es mayor que ”ciego“

Pero miremos lo siguiente:

- “Casa” es MAYOR que “fuego”
- “cielo” es MENOR que “Ciego”

Las mayúsculas se consideran “más grandes” que las minúsculas...

Python utiliza un criterio de comparación de cadenas muy natural: el orden alfabético

Observemos ...

- “casa“ es menor que ”fuego“
- ”cielo“ es mayor que ”ciego“

Pero miremos lo siguiente:

- “Casa” es MAYOR que “fuego”
- “cielo” es MENOR que “Ciego”

Las mayúsculas se consideran “más grandes” que las minúsculas...

Los caracteres se representan usando una codificación que los asocian a números. ¿Conocen la codificación ASCII?

Existe una función que me retorna esa codificación: **ord()**

Ejemplo

- ord('a') => 97
- ord('N') => 78

Por lo tanto: 'N' < 'a'

Los caracteres se representan usando una codificación que los asocian a números. ¿Conocen la codificación ASCII?
Existe una función que me retorna esa codificación: **ord()**

Ejemplo

- ord('a') => 97
- ord('N') => 78

Por lo tanto: 'N' < 'a'

Los caracteres se representan usando una codificación que los asocian a números. ¿Conocen la codificación ASCII?
Existe una función que me retorna esa codificación: **ord()**

Ejemplo

- ord('a') => 97
- ord('N') => 78

Por lo tanto: 'N' < 'a'

Trabajando con cadenas de caracteres

Separando cadenas: La función split

split es una función que devuelve una lista con todas las palabras de la cadena usando como separador al carácter pasado como parámetro.

```
>>> cadena = "hola-como-estas"
>>> cadena.split('-')
>>> ['hola','como','estas']
```

Separador: -

```
>>> cadena = "hola como estas"
>>> cadena.split(' ')
>>> ['hola','como','estas']
```

Separador: **espacio**

Trabajando con cadenas de caracteres

Separando cadenas: La función split

split es una función que devuelve una lista con todas las palabras de la cadena usando como separador al carácter pasado como parámetro.

```
>>> cadena = "hola-como-estas"
>>> cadena.split('-')
>>> ['hola','como','estas']
```

Separador: -

```
>>> cadena = "hola como estas"
>>> cadena.split(' ')
>>> ['hola','como','estas']
```

Separador: **espacio**

Trabajando con cadenas de caracteres

Modificando las cadenas: Las funciones lstrip y rstrip

Estas funciones permiten eliminar ciertos caracteres ya sea se encuentren a la izquierda o a la derecha de la cadena de caracteres..

lstrip elimina caracteres de la **izquierda**. En este caso espacio.

```
>>> cadena = '   espacios  '
>>> cadena.lstrip()
>>> 'espacios  '
```

rstrip elimina caracteres de la **derecha**. En este caso "X".

```
>>> cadena = '   espaciosXXXXXX'
>>> cadena.rstrip("X")
>>> '   espacios'
```

Trabajando con cadenas de caracteres

Modificando las cadenas: Las funciones lstrip y rstrip

Estas funciones permiten eliminar ciertos caracteres ya sea se encuentren a la izquierda o a la derecha de la cadena de caracteres..

lstrip elimina caracteres de la **izquierda**. En este caso espacio.

```
>>> cadena = '   espacios  '
>>> cadena.lstrip()
>>> 'espacios  '
```

rstrip elimina caracteres de la **derecha**. En este caso "X".

```
>>> cadena = '   espaciosXXXXXX'
>>> cadena.rstrip("X")
>>> '   espacios'
```

Trabajando con cadenas de caracteres

Mayúsculas y minúsculas

Es posible convertir de mayúsculas a minúsculas y viceversa

Convertir a mayúsculas

```
cadena = 'espacios'
cadena.upper()
'ESPACIOS'
```

Convertir a minúsculas

```
cadena = 'ESPACIOS'
cadena.lower()
'espacios'
```

Trabajando con Listas

Definición y uso

Las **listas** son un tipo de datos provisto por Python

- Colección ordenada de datos.
- Cualquier tipo de datos.
- Tamaño cambiante.

Ejemplos

lista = []
lista = [1,2,3]
lista = [1, "Hola"]

Trabajando con Listas

Algunas funciones

Comando	**Descripción**	**Ejemplo**
append(x)	Agrega un ítem al final	lista.append(5)
insert(i, x)	Inserta un ítem en una posición dada	lista.insert(3,7)
remove(x)	Elimina el primer ítem de la lista cuyo valor sea igual a x	lista.remove(5)
index(x)	Devuelve el índice en la lista del primer ítem cuyo valor sea x	lista.index(5)
count(x)	Devuelve el número de veces que aparece el ítem	lista.count(10)

Trabajando con Listas

Listas y parámetros ...

Modificación de las listas.

```
def modifica1(x):
    x = [1,2,3]
    print x
a=['a','b']
modifica1(a)
print a
```

salida

```
>>>
[1, 2, 3]
['a', 'b']
>>>
```

```
def modifica2(x):
    x[0] = [1,2,3]
    print x
a=['a','b']
modifica2(a)
print a
```

salida

```
>>>
[[1, 2, 3] ,'b']
[[1, 2, 3] ,'b']
>>>
```

? **¿Qué pasó?**

Trabajando con Listas

Listas y parámetros ...

Modificación de las listas.

```
def modifica1(x):
    x = [1,2,3]
    print x
a=['a','b']
modifica1(a)
print a
```

salida

```
>>>
[1, 2, 3]
['a', 'b']
>>>
```

```
def modifica2(x):
    x[0] = [1,2,3]
    print x
a=['a','b']
modifica2(a)
print a
```

salida

```
>>>
[[1, 2, 3] ,'b']
[[1, 2, 3] ,'b']
>>>
```

? ¿Qué pasó?

Trabajando con Listas

Listas y parámetros ...

Modificación de las listas.

```
def modifica1(x):
    x = [1,2,3]
    print x
a=['a','b']
modifica1(a)
print a
```

salida

```
>>>
[1, 2, 3]
['a', 'b']
>>>
```

```
def modifica2(x):
    x[0] = [1,2,3]
    print x
a=['a','b']
modifica2(a)
print a
```

salida

```
>>>
[[1, 2, 3] ,'b']
[[1, 2, 3] ,'b']
>>>
```

? ¿Qué pasó?

Trabajando con Listas

Listas y parámetros ...

Modificación de las listas.

```
def modifica1(x):
    x = [1,2,3]
    print x
a=['a','b']
modifica1(a)
print a
```

salida

```
>>>
[1, 2, 3]
['a', 'b']
>>>
```

```
def modifica2(x):
    x[0] = [1,2,3]
    print x
a=['a','b']
modifica2(a)
print a
```

salida

```
>>>
[[1, 2, 3] ,'b']
[[1, 2, 3] ,'b']
>>>
```

? ¿Qué pasó?

Trabajando con Listas

Listas y parámetros ...

Modificación de las listas.

```
def modifica1(x):
    x = [1,2,3]
    print x
a=['a','b']
modifica1(a)
print a
```

salida

```
>>>
[1, 2, 3]
['a', 'b']
>>>
```

```
def modifica2(x):
    x[0] = [1,2,3]
    print x
a=['a','b']
modifica2(a)
print a
```

salida

```
>>>
[[1, 2, 3] ,'b']
[[1, 2, 3] ,'b']
>>>
```

? ¿Qué pasó?

Trabajando con Tuplas

Definición y uso

Las **tuplas** también son tipos de datos provistos por Python

Ejemplos

tupla =1,2
tupla =(1,2)

A diferencia de las listas, son **INMUTABLES**. Es decir, que no se pueden modificar.

Permitido

```
u = tupla, 3
```

No permitido

```
tupla[0]=77
```

Trabajando con Tuplas

Definición y uso

Las **tuplas** también son tipos de datos provistos por Python

Ejemplos

tupla =1,2
tupla =(1,2)

A diferencia de las listas, son **INMUTABLES**. Es decir, que no se pueden modificar.

Permitido

u = tupla, 3

No permitido

tupla[0]=77

Trabajando con archivos

Definición

Archivo

Es una estructura que me permite almacenar datos en una memoria externa.

Estos datos **permanecen** en la computadora cuando nuestro programa termina su ejecución.

Trabajando con archivos

Definición

Archivo

Es una estructura que me permite almacenar datos en una memoria externa.
Estos datos **permanecen** en la computadora cuando nuestro programa termina su ejecución.

Trabajando con archivos

Operaciones típicas

Por lo general,

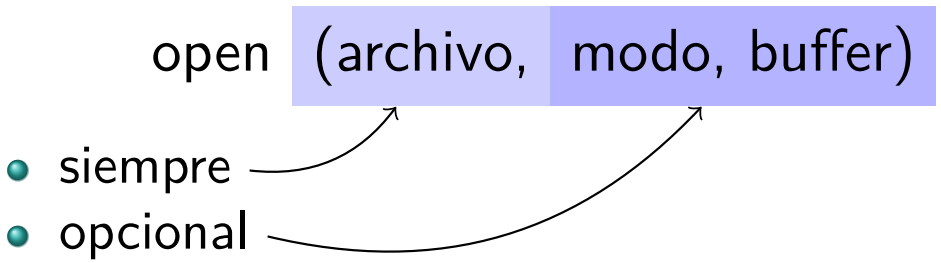
- Abrimos el archivo
- Leemos o grabamos datos
- Cerramos el archivo

Trabajando con archivos

¿Cómo abrimos un archivo?

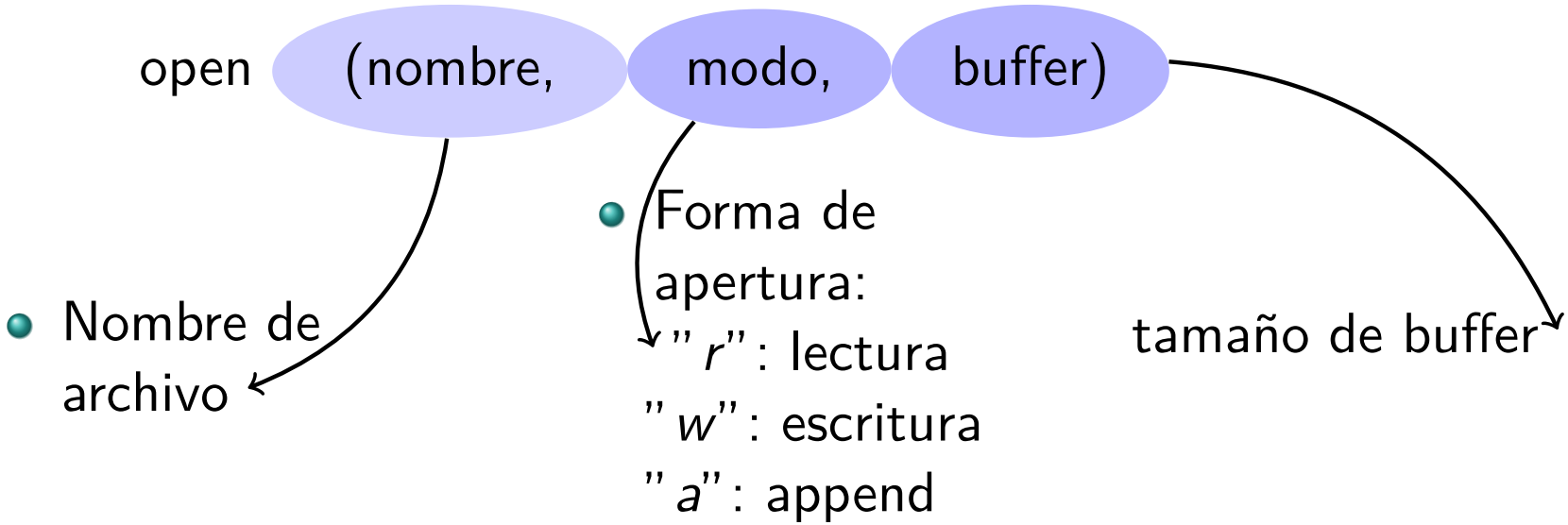
Los archivos se crean usando la función **open()**

- Abrir un archivo: open('nombre')
- Opciones



Trabajando con archivos

Abrir archivos



Trabajando con archivos

¿Cómo abrimos un archivo?

- archi=open("datos.txt", "r"): Esto abre el arhivo denominado "datos.txt" para leerlo
- archi=open("datos.txt", "a"): Esto abre el arhivo denominado "datos.txt" para agregar datos
- archi=open("datos.txt", "w"): Esto abre el arhivo denominado "datos.txt" para escribirlo

¿Qué pasa si datos.txt no existe?

Trabajando con archivos

¿Cómo abrimos un archivo?

- archi=open("datos.txt", "r"): Esto abre el arhivo denominado "datos.txt" para leerlo
- archi=open("datos.txt", "a"): Esto abre el arhivo denominado "datos.txt" para agregar datos
- archi=open("datos.txt", "w"): Esto abre el arhivo denominado "datos.txt" para escribirlo

¿Qué pasa si datos.txt no existe?

Trabajando con archivos

¿Cómo abrimos un archivo?

- archi=open("datos.txt", "r"): Esto abre el arhivo denominado "datos.txt" para leerlo
- archi=open("datos.txt", "a"): Esto abre el arhivo denominado "datos.txt" para agregar datos
- archi=open("datos.txt", "w"): Esto abre el arhivo denominado "datos.txt" para escribirlo

¿Qué pasa si datos.txt no existe?

Trabajando con archivos

¿Cómo abrimos un archivo?

- archi=open("datos.txt", "r"): Esto abre el arhivo denominado "datos.txt" para leerlo
- archi=open("datos.txt", "a"): Esto abre el arhivo denominado "datos.txt" para agregar datos
- archi=open("datos.txt", "w"): Esto abre el arhivo denominado "datos.txt" para escribirlo

¿Qué pasa si datos.txt no existe?

Trabajando con archivos

¿Cómo abrimos un archivo?

- archi=open(“datos.txt”, “r”): Esto abre el arhivo denominado “datos.txt” para leerlo
- archi=open(“datos.txt”, “a”): Esto abre el arhivo denominado “datos.txt” para agregar datos
- archi=open(“datos.txt”, “w”): Esto abre el arhivo denominado “datos.txt” para escribirlo

¿Qué pasa si datos.txt no existe?

Trabajando con archivos

¿Cómo abrimos un archivo?

- archi=open(“datos.txt”, “r”): Esto abre el arhivo denominado “datos.txt” para leerlo
- archi=open(“datos.txt”, “a”): Esto abre el arhivo denominado “datos.txt” para agregar datos
- archi=open(“datos.txt”, “w”): Esto abre el arhivo denominado “datos.txt” para escribirlo

¿Qué pasa si datos.txt no existe?

Trabajando con archivos

¿Cómo abrimos un archivo?

- archi=open(“datos.txt”, “r”): Esto abre el arhivo denominado “datos.txt” para leerlo
- archi=open(“datos.txt”, “a”): Esto abre el arhivo denominado “datos.txt” para agregar datos
- archi=open(“datos.txt”, “w”): Esto abre el arhivo denominado “datos.txt” para escribirlo

¿Qué pasa si datos.txt no existe?

Trabajando con archivos

Lectura y Escritura

Métodos read() y write()

```
>>> a = open('archivo.txt', 'w')
>>> a.write('Hola, ')
>>> a.write('Mundo!')
>>> a.close()
>>> a = open('archivo.txt', 'r')
>>> a.read(4)
'Hola'
>>> a.read()
', Mundo!'
```

- a.close()
  Cierra el archivo
- a.write(cadena)
  Escribe la cadena en el archivo
- a.read(cantidadBytes)
  - Si **cantidadBytes** es < 0 o no está, lee hasta fin de archivo
  - Retorna " " si EOF

Métodos read() y write()

```
>>> a = open('archivo.txt', 'w')
>>> a.write('Hola, ')
>>> a.write('Mundo!')
>>> a.close()
>>> a = open('archivo.txt', 'r')
>>> a.read(4)
'Hola'
>>> a.read()
', Mundo!'
```

- a.close()
  Cierra el archivo
- a.write(cadena)
  Escribe la cadena en el archivo
- a.read(cantidadBytes)
  - Si **cantidadBytes** es < 0 o no está, lee hasta fin de archivo
  - Retorna " " si EOF

Trabajando con archivos

Lectura y Escritura

Métodos read() y write()

```
>>> a = open('archivo.txt', 'w')
>>> a.write('Hola, ')
>>> a.write('Mundo!')
>>> a.close()
>>> a = open('archivo.txt', 'r')
>>> a.read(4)
'Hola'
>>> a.read()
', Mundo!'
```

- a.close()
  Cierra el archivo
- a.write(cadena)
  Escribe la cadena en el archivo
- a.read(cantidadBytes)
  - Si **cantidadBytes** es < 0 o no está, lee hasta fin de archivo
  - Retorna " " si EOF

Métodos read() y write()

```
>>> a = open('archivo.txt', 'w')
>>> a.write('Hola, ')
>>> a.write('Mundo!')
>>> a.close()
>>> a = open('archivo.txt', 'r')
>>> a.read(4)
'Hola'
>>> a.read()
', Mundo!'
```

- a.close()
  Cierra el archivo
- a.write(cadena)
  Escribe la cadena en el archivo
- a.read(cantidadBytes)
  - Si **cantidadBytes** es < 0 o no está, lee hasta fin de archivo
  - Retorna " " si EOF

Trabajando con archivos

Lectura y Escritura

Método writelines()

```
lista = [“uno”,“dos”,“tres”]
i=0
for x in lista:
  lista[i] = x+’\n’
  i = i+1
archi = open(“archivo.txt”, “w”)
archi.writelines(lista)
archi.close()
```

Fin de línea

Método writelines()

```
lista = [“uno”,“dos”,“tres”]
i=0
for x in lista:
  lista[i] = x+’\n’
  i = i+1
archi = open(“archivo.txt”, “w”)
archi.writelines(lista)
archi.close()
```

Fin de línea

Método writelines()

```
lista = [“uno”,“dos”,“tres”]
i=0
for x in lista:
  lista[i] = x+’\n’
  i = i+1
archi = open(“archivo.txt”, “w”)
archi.writelines(lista)
archi.close()
```

Fin de línea

Trabajando con archivos

Lectura y Escritura

Método writelines()

```
lista = [“uno”,“dos”,“tres”]
i=0
for x in lista:
  lista[i] = x+’\n’
  i = i+1
archi = open(“archivo.txt”, “w”)
archi.writelines(lista)
archi.close()
```

Fin de línea

writelines() no agrega por defecto el fin de línea.

Métodos readlines() y readline()

```
>>> f = open("lineas.txt","r")
>>> f.readlines()
['uno\n','dos\n', 'tres\n']
```

readline(): lee una línea.
readlines(): retorna una lista con las líneas.

Trabajando con archivos

Lectura y Escritura

Métodos readlines() y readline()

```
>>> f = open("lineas.txt","r")
>>> f.readlines()
['uno\n','dos\n', 'tres\n']
```

readline(): lee una línea.
readlines(): retorna una lista con las líneas.

Trabajando con archivos

Lectura y Escritura

Alternativa a **readlines()**

```
>>> f = open("lineas.txt","r")
>>> f.readlines()
['uno\n','dos\n', 'tres\n']
```

Leer por línea

```
>>> f = open("lineas.txt","r")
>>> for linea in f:
        print linea
uno
dos
tres
```

Trabajando con archivos

Agregar

Método para agregar al final datos al archivo.

- Abrimos en modo **append**.

```
>>> f = open("lineas.txt","a")
```

- Agregamos al archivo con a.writelines(lista).

```
lista = ["cuatro","cinco","seis"]
i=0
for x in lista:
  lista[i] = x+'\\n'
  i = i+1
archi = open("archivo.txt", "a")
archi.writelines(lista)
archi.close()
```

Trabajando con archivos

Agregar

Método para agregar al final datos al archivo.

- Abrimos en modo **append**.

```
>>> f = open("lineas.txt","a")
```

- Agregamos al archivo con a.writelines(lista).

```
lista = ["cuatro","cinco","seis"]
i=0
for x in lista:
  lista[i] = x+'\\n'
  i = i+1
archi = open("archivo.txt", "a")
archi.writelines(lista)
archi.close()
```

Trabajando con archivos

Lectura y escritura

Para abrir un archivo combinando las acciones de lectura y escritura.

- Abrimos en modo **lectura-escritura**.

```
archi = open("archivo.txt", "r+")
for linea in archi:
          print linea
lista = ["uno","dos","tres"]
i=0
for x in lista:
  lista[i] = x+'\n'
  i = i+1

archi.writelines(lista)
archi.close()
```

Trabajando con archivos

Moviéndose por el archivo

Método para movernos a.tell() a.seek().

- a.tell() devuelve un entero que indica la posición actual en el archivo medida en bytes.
- a.seek(desplazamiento, desde_donde) te ubica en la posición indicada en los argumentos.
  - a.seek(8) va al bayte 9 del archivo.
  - a.seek(8, 0) se mueve 8 bytes desde el comienzo del archivo.
  - a.seek(8, 1) se mueve 8 bytes desde donde está actualmente.
  - a.seek(-8, 2) se mueve a 8 bytes antes del final del archivo.

Trabajando con archivos

Moviéndose por el archivo

Método para movernos a.tell() a.seek().

- a.tell() devuelve un entero que indica la posición actual en el archivo medida en bytes.
- a.seek(desplazamiento, desde_donde) te ubica en la posición indicada en los argumentos.
  - a.seek(8) va al bayte 9 del archivo.
  - a.seek(8, 0) se mueve 8 bytes desde el comienzo del archivo.
  - a.seek(8, 1) se mueve 8 bytes desde donde está actualmente.
  - a.seek(-8, 2) se mueve a 8 bytes antes del final del archivo.

Ahora si..

A trabajar!! ..