1.1 Python

¿Qué es Python?

Python es un lenguaje interpretado de alto nivel. Frecuentemente se lo clasifica como lenguaje de "scripting" . La sintaxis del Python tiene elementos de lenguaje C de programación.

Python fue creado por Guido van Rossum a principios de la década del '90 y lo nombró así en honor de Monty Python.

¿Dónde conseguir Python?

Te recomendamos instalar Python 3.6 o más nuevo. En la documentación previa hablamos sobre cómo instalar Python para este curso .

¿Para qué fue creado Python?

El objetivo original de su autor fue crear un lenguaje de programación con el que podría realizar las tareas de administración de un sistema fácilmente. En algún sentido los scripts de la terminal no eran suficientemente poderosos y programar esas tareas en C resultaba demasiado tedioso. Python fue creado para llenar ese hueco en el medio.

¿Cómo ejecuto Python en mi máquina?

Existen diferentes entornos en los que podés correr Python en tu computadora. Es importante saber que Python está instalado normalmente como un programa que se ejecuta desde la consola. Desde la terminal deberías poder ejecutar pythonasí:

bash $ python

Python 3.8.1 (default, Feb 20 2020, 09:29:22)

[Clang 10.0.0 (clang-1000.10.44.4)] on darwin

Type "help", "copyright", "credits" or "license" for more information.

>>> print("hello world")

hello world

>>>

Si es la primera vez que ves una consola o terminal, sería conveniente que pares aquí, leas un tutorial corto sobre cómo usar la consola de tu sistema operativo y luego vuelvas para seguir leyendo.

Existen diversos entornos fuera de la terminal en los que se puede escribir y ejecutar código Python. Pero para nosotros es importante que primero aprendas a usarlo desde la terminal: si lo sabés usar bien desde la terminal (que es su entorno natural) lo podrás usar en cualquier otro entorno. Ya en la próxima clase usarás Python dentro de un entorno de desarrollo. Por ahora, te recomendamos usarlo de esta manera que acabamos de explicar.

Ejercicios

Ejercicio 1.1: Python como calculadora

En tu máquina, iniciá Python y usalo como calculadora para resolver el siguiente problema:

¿Cuántas horas son 105 minutos?

¿Cuántos kilómetros son 20 millas? (un kilómetro corresponde a 0,6214 millas)

>> > 105 / 60

1,75

>> > 20 / 0.6214

32,1853878339234

consejo: Usá el guión bajo (guión bajo, \_) para referirte al resultado del último cálculo.

Si alguien corre una carrera de 20 millas en 105 minutos, ¿cuál fue tu velocidad promedio en km / h?

>> > \_ / 1,75

18,391650190813372

Ejercicio 1.2: Obtener ayuda

Utilice el comando help()para obtener ayuda sobre la función abs(). Luego, usá el help()para obtener la ayuda sobre la función round(). Tipeá help()sólo para entrar en la ayuda en modo interactivo.

El help()no Funciona con los Comandos Básicos de Python de Como for, if, while, etc. Si tipeás help(for)Vas A Obtener error ONU. Podés probar usando comillas como en help("for"), en algunos entornos funciona bien. Si no, siempre podés hacer una búsqueda en internet.

La documentación oficial en inglés de Python se encuentra en http://docs.python.org . Por ejemplo, encontrá ahí la documentación sobre la función abs()(ayuda: está dentro de "library reference" y relacionado con las "built-in functions").

Ejercicio 1.3: Copiar y pegar

Este curso está estructurado como una serie de páginas web tradicionales en las que les incentivamos a probar interactivamente fragmentos de código en sus intérpretes de Python escribiéndolos a mano . Si estás aprendiendo Python por primera vez, esta forma "lenta" de hacer las cosas es la que recomendamos. Vas a entender mejor yendo lento y escribiendo los comandos vos mismo mientras pensás en lo que estás tipeando.

Es importante que tipées los comandos a mano. Para usar copiar-pegar quizás mejor ni hacerlos. Parte del objetivo de los ejercicios es entrenar tus manos, tus ojos y tu cabeza en leer, escribir y mirar código tal como dice Zed Shaw en su libro . Usar copiar-pegar excesivamente es como hacerte trampa a vos misme. Es como tratar de aprender a tocar la guitarra escuchando discos: es probable que no aprendas nunca.

Si, de todas formas, en algún momento necesitás hacer "copiar y pegar" de fragmentos de código, seleccionar el código que luego viene del símbolo >>>y hasta la siguiente línea en blanco o el siguiente >>>(el que aparece primero). Seleccioná "copiar" en el navegador (Ctrl-C), y al intérprete de Python y poné "pegar" (Ctrl-V o Crtl-shift-V) para pegarlo. Para ejecutar el código es posible que tengas que apretar "Enter" luego de pegarlo.

Usá copiar y pegar para ejecutar los siguientes comandos:

>> > 12 + 20

32

>> > ( 3 + 4

+ 5 + 6 )

18

>> > para i en gama ( 5 ):

de impresión ( i )

0

1

2

3

4

>> >

1.2 Un primer programa

En esta sección vas a crear tu primer programa en Python, ejecutarlo y debuguearlo.

Ejecutando Python

Los programas en Python siempre son ejecutados en un intérprete de Python.

El intérprete es una aplicación que funciona en la consola y se ejecuta desde la terminal.

python3

Python 3.6.1 (v3.6.1:69c0db5050, Mar 21 2017, 01:21:04)

[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin

Type "help", "copyright", "credits" or "license" for more information.

>>>

Les programadores no suelen tener problemas en usar el intérprete de esta forma, aunque no es la más cómoda para principiantes. Más adelante vamos a proponerles usar entornos de desarrollo más sofisticados, pero por el momento quedémosnos con la incomodidad que nos va a enseñar cosas útiles.

Modo interactivo

Cuando ejecutás Python, entrás al modo interactivo en el que podés experimentar.

Si escribís un comando, se va a ejecutar inmediatamente. No hay ningún ciclo de edición-compilación-ejecución-debug en Python, como hay en otros lenguajes.

>>> print('hello world')

hello world

>>> 37\*42

1554

>>> for i in range(5):

... print(i)

...

0

1

2

3

4

>>>

Esta forma de escribir código (en una consola del lenguaje) que se evalúa inmediatamente e imprime el resultado, se denomina bucle de Lectura-Evaluación-Impresión (REPL por las siglas en inglés de «Read-Eval-Print-Loop»). Asegurate de poder interactuar con el intérprete antes de seguir.

Veamos en mayor detalle cómo funciona este REPL:

>>> es el símbolo del intérprete para comenzar un nuevo comando.

... es el símbolo del intérprete para continuar con un comando comenzado antes. Dejá una línea en blanco para terminar lo que ya ingresaste.

El símbolo ... puede mostrarse o no dependiendo de tu entorno. En este curso lo mostraremos como líneas en blanco para facilitar el copy-paste de fragmentos de código (del que ya dijimos, ¡no hay que abusar!).

Antes vimos que el guión bajo \_ guarda el último resultado.

>>> 37 \* 42

1554

>>> \_ \* 2

3108

>>> \_ + 50

3158

>>>

Esto solo es válido en el modo interactivo que estamos viendo. No uses el guión bajo en un programa.

Crear programas

Los programas se guardan en archivos .py.

# hello.py

print('hello world')

Podés crear estos archivos con tu editor de texto favorito. Más adelante vamos a proponerles usar el spyder que es un entorno de desarrollo integrado (IDE por «Integrated Development Environment», entorno de desarrollo integrado) que permite tener en la pantalla un editor y un intérprete al mismo tiempo, entre otras cosas. Pero por ahora usemos el block de notas, el gedit o tu editor favorito para seguir estos ejemplos.

Ejecutar programas

Para ejecutar un programa, correlo en la terminal con el comando python seguido del nombre del archivo a ejecutar. Por ejemplo, en una línea de comandos Unix (por ejemplo Ubuntu):

bash % python hello.py

hello world

bash %

O en una terminal de Windows:

C:\SomeFolder>hello.py

hello world

C:\SomeFolder>c:\python36\python hello.py

hello world

Obervación: En Windows puede ser necesario especificar el camino (path) completo al intérprete de Python como en c:\python36\python. Sin embargo, si Python está instalado del modo usual, podría alcanzar con que tipées el nombre del programa como en hello.py.

Tené en cuenta que con estos comandos estás corriendo el código de Python desde la línea de comandos de tu sistema operativo. El código se ejecuta, Python termina y el control vuelve a la terminal, saliendo de Python. Si necesitás ejecutarlo y seguir dentro del intérprete de Python podés usar python -i hello.py.

Si estás dentro del intérprete de Python y querés salir y volver a la línea de comandos, podés hacerlo mediante el comando exit().

Un ejemplo de programa

Resolvamos el siguiente problema:

Una mañana ponés un billete en la vereda al lado del obelisco porteño. A partir de ahí, cada día vas y duplicás la cantidad de billetes, apilándolos prolijamente. ¿Cuánto tiempo pasa antes de que la pila de billetes sea más alta que el obelisco?

Acá va una solución:

# obelisco.py

grosor\_billete = 0.11 \* 0.001 # grosor de un billete en metros

altura\_obelisco = 67.5 # altura en metros

num\_billetes = 1

dia = 1

while num\_billetes \* grosor\_billete <= altura\_obelisco:

print(dia, num\_billetes, num\_billetes \* grosor\_billete)

dia = dia + 1

num\_billetes = num\_billetes \* 2

print('Cantidad de días', dia)

print('Cantidad de billetes', num\_billetes)

print('Altura final', num\_billetes \* grosor\_billete)

Cuando lo ejecutás, la salida será la siguiente:

bash % python3 obelisco.py

1 1 0.00011

2 2 0.00022

3 4 0.00044

4 8 0.00088

5 16 0.00176

6 32 0.00352

...

19 262144 28.83584

20 524288 57.67168

Cantidad de días 21

Cantidad de billetes 1048576

Altura final 115.34336

A continuación vamos a usar este primer programa como ejemplo para aprender algunas cosas fundamentales sobre Python.

Comandos

Un programa de Python es una secuencia de comandos:

a = 3 + 4

b = a \* 2

print(b)

Cada comando se termina con una nueva línea. Los comandos son ejecutados uno luego del otro hasta que el intérprete llega al final del archivo.

Comentarios

Los comentarios son texto que no será ejecutado.

a = 3 + 4

# Esto es un comentario

b = a \* 2

print(b)

Los comentarios comienzan con # y siguen hasta el final de la línea.

Variables

Una variable es un nombre para un valor. Estos nombres pueden estar formados por letras (minúsculas y mayúsculas) de la a a la z. También pueden incluir el guión bajo, y se pueden usar números, salvo como primer caracter.

altura = 442 # válido

\_altura = 442 # válido

altura2 = 442 # válido

2altura = 442 # inválido

Tipos

El tipo de las variables no debe ser declarado como en otros lenguajes. El tipo es asociado con el valor del lado derecho.

altura = 442 # Entero

altura = 442.0 # Punto flotante

altura = 'Muy, muy alto' # Cadena de caracteres

Decimos que Python tiene tipado dinámico. El tipo percibido por el intérprete puede cambiar a lo largo de la ejecución dependiendo del valor asignado a la variable.

Python distingue mayúsculas y minúsculas

Mayúsculas y minúsculas son diferentes para Python. Por ejemplo, todas las siguientes variables son diferentes.

nombre = 'David'

Nombre = 'Diego'

NOMBRE = 'Rosita'

Los comandos de Python siempre se escriben con minúsculas.

while x < 0: # OK

WHILE x < 0: # ERROR

Ciclos

El comando while ejecuta un ciclo o loop.

while num\_billetes \* grosor\_billete <= altura\_obelisco:

print(dia, num\_billetes, num\_billetes \* grosor\_billete)

dia = dia + 1

num\_billetes = num\_billetes \* 2

print('Cantidad de días', dia)

Los comandos indentados debajo del while se van a a ejecutar mientras que la expresión luego del while sea verdadera (True).

Indentación

La indentación se usa para marcar grupos de comandos que van juntos. Considerá el ejemplo anterior:

while num\_billetes \* grosor\_billete <= altura\_obelisco:

print(dia, num\_billetes, num\_billetes \* grosor\_billete)

dia = dia + 1

num\_billetes = num\_billetes \* 2

print('Cantidad de días', dia)

La indentación agrupa los comandos siguientes como las operaciones a repetir:

print(dia, num\_billetes, num\_billetes \* grosor\_billete)

dia = dia + 1

num\_billetes = num\_billetes \* 2

Como el comando print() del final no está indentado, no pertenece al ciclo. La línea en blanco que dejamos entre ambos solo está para facilitar la lectura y no afecta la ejecución.

Indentando adecuadamente

Algunas recomendaciones sobre cómo indentar:

Usá espacios y no el tabulador.

Usá 4 espacios por cada nivel.

Usá un editor de textos que entienda que estás escribiendo en Python.

El único requisito del intérprete de Python es que la indentación dentro de un mismo bloque sea consistente. Por ejemplo, esto es un error:

while num\_billetes \* grosor\_billete <= altura\_obelisco:

print(dia, num\_billetes, num\_billetes \* grosor\_billete)

dia = dia + 1 # ERROR

num\_billetes = num\_billetes \* 2

Condicionales

El comando if es usado para ejecutar un condicional:

if a > b:

print('Gana a')

else:

print('Gana b')

Podés verificar condiciones mutuamente excluyentes agregando condiciones extras con elif.

if a > b:

print('Gana a')

elif a == b:

print('Empate!')

else:

print('Gana b')

El comando elif viene de else, if y puede traducirse como "si no se da la condición del if anterior, verificá si se da la siguiente".

Imprimir en pantalla

La función print imprime una línea de texto con el valor pasado como parámetro.

print('Hello world!') # Imprime 'Hello world!'

Podés imprimir variables. El texto impreso en ese caso será el valor de la variable y no su nombre.

x = 100

print(x) # imprime el texto '100'

Si le pasás más de un valor al print los separa con espacios.

nombre = 'Juana'

print('Mi nombre es', nombre) # Imprime el texto 'Mi nombre es Juana'

print() siempre termina la línea impresa pasando a la siguiente.

print('Hola')

print('Mi nombre es', 'Juana')

Esto imprime:

Hola

Mi nombre es Juana

El salto de línea entre ambos comandos puede ser suprimido:

print('Hola', end=' ')

print('Mi nombre es', 'Juana')

Este código va a imprimir:

Hola Mi nombre es Juana

Ingreso de valores por teclado

Para leer un valor ingresado por el usuario, usá la función input():

nombre = input('Ingresá tu nombre:')

print('Tu nombre es', nombre)

input imprime el texto que le pases como parámetro y espera una respuesta. Es útil para programas pequeños, para hacer ejercicios o para debuguear un código. Casi no se lo usa en programas reales.

El comando pass

A veces es conveniente especificar un bloque de código que no haga nada. El comando pass se usa para eso.

if a > b:

pass

else:

print('No ganó a')

Este comando no hace nada. Sirve para guardar el lugar para un comando que querramos agregar luego.

Ejercicios

Ejercicio 1.4: Debuguear

El siguiente fragmento de código está relacionado con el problema del obelisco. Tiene un bug, es decir, un error.

# obelisco.py

grosor\_billete = 0.11 \* 0.001 # 0.11 mm en metros

altura\_obelisco = 67.5 # altura en metros

num\_billetes = 1

dia = 1

while num\_billetes \* grosor\_billete <= altura\_obelisco:

print(dia, num\_billetes, num\_billetes \* grosor\_billete)

dia = dias + 1

num\_billetes = num\_billetes \* 2

print('Cantidad de días', dia)

print('Cantidad de billetes', num\_billetes)

print('Altura final', num\_billetes \* grosor\_billete)

Copiá y pegá el código que aparece arriba en un nuevo archivo llamado obelisco.py. Cuando ejecutes el código vas a obtener el siguiente mensaje de error que hace que el programa se detenga:

Traceback (most recent call last):

File "obelisco.py", line 10, in <module>

dia = dias + 1

NameError: name 'dias' is not defined

Aprender a leer y entender los mensajes de error es una parte fundamental de programar en Python. Si tu programa crashea (se rompe, da error) la última línea del mensaje de error indica el motivo. Un poco más arriba vas a ver un fragmento de código, un nombre de archivo y un número de línea que identifican el problema.

¿En qué linea está el error?

¿Cuál es el error?

Repará el error.

Ejecutá el programa exitosamente.

Ejercicio 1.5: La pelota que rebota

Este es el primer conjunto de ejercicios en el que vas a tener que crear un archivo de Python y correrlo. A partir de aca, vamos a asumir que estás trabajando en el subdirectorio ejercicios\_python/. Para ayudarte a organizar los archivos de diferentes clases y a ubicar el lugar correcto ya creamos algunos subdirectorios y un par de archivos en el directorio correpondiente a esta clase. Buscá en tu terminal el archivo ejercicios\_python/Clase01/rebotes.py (cambiando de directorio como vimos recién). Lo vamos a usar en este ejercicio.

Una pelota de goma es arrojada desde una altura de 100 metros y cada vez que toca el piso salta 3/5 de la altura desde la que cayó. Escribí un programa rebotes.py que imprima una tabla mostrando las alturas que alcanza en cada uno de sus primeros diez rebotes.

Tu programa debería generar una tabla que se parezca a esta:

1 60.0

2 36.0

3 21.599999999999998

4 12.959999999999999

5 7.775999999999999

6 4.6655999999999995

7 2.7993599999999996

8 1.6796159999999998

9 1.0077695999999998

10 0.6046617599999998

Nota: Podés limpiar un toque la salida si usás la función round() de la que miraste el help hace un rato. Tratá de usarla para redondear a cuatro dígitos.

1 60.0

2 36.0

3 21.6

4 12.96

5 7.776

6 4.6656

7 2.7994

8 1.6796

9 1.0078

10 0.6047

Ejercicio 1.6: Saludos

Escribí un programa llamado saludo.py que pregunte el nombre de le usuarie, imprima un saludo (por ejemplo, "Hola, Juana") y termine.

1.3 Números

Esta sección introduce las operaciones matemáticas elementales.

Tipos de números

Python tiene 4 tipos de números:

Booleanos

Enteros

Punto flotante

Complejos (con parte real y parte imaginaria)

Booleanos (bool)

Las variables booleanas se llaman así en honor al lógico inglés George Boole. Pueden tomar dos valores: True y False (verdadero y falso).

a = True

b = False

Internamente, son evaluados como enteros con valores 1, 0.

c = 4 + True # 5

d = False

if d == 0:

print('d is False')

No escribas código basado en esta convención. Sería bastante raro.

Enteros (int)

Representan números enteros (positivos y negativos) de cualquier magnitud:

a = 37

b = -299392993727716627377128481812241231

Incluso se pueden especificar en diferentes bases:

c = 0x7fa8 # Hexadecimal

d = 0o253 # Octal

e = 0b10001111 # Binario

Operaciones usuales:

x + y Suma

x - y Resta

x \* y Multiplicación

x / y División (da un float, no un int)

x // y División entera (da un int)

x % y Módulo (resto)

x \*\* y Potencia

abs(x) Valor absoluto

La unidad mínima de almacenamiento de una computadora es un bit, que puede valer 0 o 1. Los números, caracteres e incluso imágenes y sonido son almacenados en la máquina usando bits. Los números enteros positivos, en particular, suelen almacenarse mediante su representación binaria (o en base dos).

Número Representación binaria

1 1

2 10

3 11

4 100

5 101

6 110

Hay algunas operaciones primitivas que se pueden hacer con los enteros a partir de su representación como bits:

x << n Desplazamiento de los bits a la izquierda

x >> n Desplazamiento de los bits a la derecha

x & y AND bit a bit.

x | y OR bit a bit.

x ^ y XOR bit a bit.

~x NOT bit a bit.

Al desplazar a la izquierda, simplemente agregamos un cero en la última posición. Así, por ejemplo si corremos el 1 dos lugares a la izquierda obtenemos un 4:

>>> 1 << 2 # 1 << 2 -> 100

4

>>> 6 & 3 # 110 & 011 -> 010

2

Al desplazar los bits de un número a la derecha un lugar, el último bit "se cae".

>>> 1 >> 1 # 1 -> 0

0

>>> 6 >> 1 # 110 -> 11

3

Punto flotante (float)

Usá una notación con decimales o una notación científica para especificar un valor de tipo punto flotante:

a = 37.45

b = 4e5 # 4 x 10\*\*5 o 400,000

c = -1.345e-10

Los números de tipo floats son representados en la máquina como números de doble precisión usando la representación nativa del microprocesador que sigue el estándar IEEE 754. Para los que los conozcan: es el mismo tipo que los double en el lenguaje C.

Un float almacenan hasta 17 digitos con un exponente entre -308 to 308

Cuidado que la aritmética de los números de punto flotante no es exacta.

>>> a = 2.1 + 4.2

>>> a == 6.3

False

>>> a

6.300000000000001

>>>

Esto no es un problema de Python, si no el resultado de la forma en que el hardware de nuestras computadoras almacena los números de punto flotante.

Operaciones usuales:

x + y Suma

x - y Resta

x \* y Multiplicación

x / y Divición (da un float, no un int)

x // y División entera (da un float, pero con ceros luego del punto)

x % y Módulo (resto)

x \*\* y Potencia

abs(x) Valor absoluto

Estas son las mismas operaciones que con los enteros. Otras operaciones usuales se encuentran en el módulo math.

import math

a = math.sqrt(x)

b = math.sin(x)

c = math.cos(x)

d = math.tan(x)

e = math.log(x)

El módulo math también tiene constantes (math.e, math.pi), entre otras cosas.

Comparaciones

Las siguientes comparaciones (suelen llamarse operadores relacionales ya que expresan una relación entre dos elementos) funcionan con números:

x < y Menor que

x <= y Menor o igual que

x > y Mayor que

x >= y Mayor o igual que

x == y Igual a

x != y No igual a

Observá que el == se usa para comparar dos elementos mientras que el = se usa para asignar un valor a una variable. Son símbolos distintos que cumplen funciones diferentes.

Podés formar expresiones booleanas más complejas usando

and, or, not

Acá mostramos algunos ejemplos:

if b >= a and b <= c:

print('b está entre a y c')

if not (b < a or b > c):

print('b sigue estando entre a y c')

Conversión de números

El nombre de un tipo (de datos) puede ser usado para convertir valores:

a = int(x) # Convertir x a int

b = float(x) # Convertir x a float

Probalo.

>>> a = 3.14159

>>> int(a)

3

>>> b = '3.14159' # También funciona con cadenas que representan números.

>>> float(b)

3.14159

>>>

Cuidado: el separador decimal en Python es el punto, como en inglés, y no la coma del castellano. Por eso el comando float(3,141592) da un ValueError.

Ejercicios

Recordatorio: Asumimos que estás trabajando en el subdirectorio ejercicios\_python/Clase01/. Buscá el archivo hipoteca.py y hacé los ejercicios en ese archivo, usando un editor de texto. Ejecutalo desde la línea de comandos.

Ejercicio 1.7: La hipoteca de David

David solicitó un crédito a 30 años para comprar una vivienda, con una tasa fija nominal anual del 5%. Pidió $500000 al banco y acordó un pago mensual fijo de $2684,11.

El siguiente es un programa que calcula el monto total que pagará David a lo largo de los años:

# hipoteca.py

saldo = 500000.0

tasa = 0.05

pago\_mensual = 2684.11

total\_pagado = 0.0

while saldo > 0:

saldo = saldo \* (1+tasa/12) - pago\_mensual

total\_pagado = total\_pagado + pago\_mensual

print('Total pagado', round(total\_pagado, 2))

Copiá este código y correlo. Deberías obtener 966279.6 como respuesta.

Ejercicio 1.8: Adelantos

Supongamos que David adelanta pagos extra de $1000/mes durante los primeros 12 meses de la hipoteca.

Modificá el programa para incorporar estos pagos extra y que imprima el monto total pagado junto con la cantidad de meses requeridos.

Cuando lo corras, este nuevo programa debería dar un pago total de 929965.62 en 342 meses.

Aclaración: aunque puede parecer sencillo, este ejercicio requiere que agregues una variable mes y que prestes bastante atención a cuándo la incrementás, con qué valor entra al ciclo y con qué valor sale del ciclo. Una posiblidad es inicializar mes en 0 y otra es inicializarla en 1. En el primer caso es problable que la variable salga del ciclo contando la cantidad de pagos que se hicieron, en el segundo, ¡es probable que salga contando la cantidad de pagos más uno!

Ejercicio 1.9: Calculadora de adelantos

¿Cuánto pagaría David si agrega $1000 por mes durante cuatro años, comenzando en el sexto año de la hipoteca (es decir, luego de 5 años)?

Modificá tu programa de forma que la información sobre pagos extras sea incorporada de manera versátil. Agregá las siguientes variables antes del ciclo, para definir el comienzo, fin y monto de los pagos extras:

pago\_extra\_mes\_comienzo = 61

pago\_extra\_mes\_fin = 108

pago\_extra = 1000

Hacé que el programa tenga en cuenta estas variables para calcular el total a pagar apropiadamente.

Ejercicio 1.10: Tablas

Modicá tu programa para que imprima una tabla mostrando el mes, el total pagado hasta el momento y el saldo restante. La salida debería verse aproximadamente así:

1 2684.11 499399.22

2 5368.22 498795.94

3 8052.33 498190.15

4 10736.44 497581.83

5 13420.55 496970.98

...

308 874705.88 3478.83

309 877389.99 809.21

310 880074.1 -1871.53

Total pagado: 880074.1

Meses: 310

Ejercicio 1.11: Bonus

Ya que estamos, corregí el código anterior de forma que el pago del último mes se ajuste a lo adeudado.

Asegurate de guardar en el archivo hipoteca.py esta última versión en tu directorio ejercicios\_python/Clase01/. Vamos a volver a trabajar con él.

Ejercicio 1.12: Un misterio

Las funciones int() y float() pueden usarse para convertir números. Por ejemplo,

>>> int("123")

123

>>> float("1.23")

1.23

>>>

Con esto en mente, ¿podrías explicar el siguiente comportamiento?

>>> bool("False")

True

>>>

Ejercicio 1.13: El volumen de una esfera

En tu directorio de trabajo de esta clase, escribí un programa llamado esfera.py que le pida al usuario que ingrese por teclado el radio r de una esfera y calcule e imprima el volumen de la misma. Sugerencia: recordar que el volúmen de una esfera es 4/3 πr^3.

Finalmente, ejecutá el programa desde la línea de comandos para responder ¿cuál es el volumen de una esfera de radio 6? Debería darte 904.7786842338603.

1.4 Cadenas

En esta sección veremos cómo trabajar con textos.

Representación de textos

Las cadenas de caracteres entre comillas se usan para representar texto en Python. En este caso, fragmentos del Martín Fierro.

# Comillas simples

a = 'Aquí me pongo a cantar, al compás de la vigüela'

# Comillas dobles

b = "Los hermanos sean unidos porque ésa es la ley primera"

# Comillas triples

c = '''

Yo no tengo en el amor

Quien me venga con querellas;

Como esas aves tan bellas

Que saltan de rama en rama

Yo hago en el trébol mi cama

Y me cubren las estrellas.

'''

Normalmente las cadenas de caracteres solo ocupan una linea. Las comillas triples nos permiten capturar todo el texto encerrado a lo largo de múltiples lineas.

No hay diferencia entre las comillas simples (') y las dobles ("). Pero el mismo tipo de comillas que se usó para abrir debe usarse para cerrar.

Código de escape

Los códigos de escape (escape codes) son expresiones que comienzan con una barra invertida, \ y se usan para representar caracteres que no pueden ser fácilmente tipeados directamente con el teclado. Estos son algunos códigos de escape usuales:

'\n' Avanzar una línea

'\r' Retorno de carro

'\t' Tabulador

'\'' Comilla literal

'\"' Comilla doble literal

'\\' Barra invertida literal

El retorno de carro (código '\r') mueve el cursor al comienzo de la línea pero sin avanzar una línea. El origen de su nombre está relacionado con las máquinas de escribir.

Representación en memoria de las cadenas

Las cadenas se representan en Python asociando a cada caracter un número entero o código Unicode. Es posible definir un caracter usando su código y códigos de escape como s = '\U0001D120' para la clave de sol.

Indexación de cadenas

Las cadenas funcionan como los vectores multidimensionales en matemática, permitiendo el acceso a los caracteres individuales. El índice comienza a contar en cero. Los índices negativos se usan para especificar una posición respecto al final de la cadena.

a = 'Hello world'

b = a[0] # 'H'

c = a[4] # 'o'

d = a[-1] # 'd' (fin de cadena)

También se puede rebanar (slice) o seleccionar subcadenas especificando un range de índices con :.

d = a[:5] # 'Hello'

e = a[6:] # 'world'

f = a[3:8] # 'lo wo'

g = a[-5:] # 'world'

El caracter que corresponde al último índice no se incluye. Si un extremo no se especifica, significa que es desde el comienzo o hasta el final, respectivamente.

Operaciones con cadenas

Concatenación, longitud, pertenencia y replicación.

# Concatenación (+)

a = 'Hello' + 'World' # 'HelloWorld'

b = 'Say ' + a # 'Say HelloWorld'

# Longitud (len)

s = 'Hello'

len(s) # 5

# Test de pertenencia (in, not in)

t = 'e' in s # True

f = 'x' in s # False

g = 'hi' not in s # True

# Replicación (s \* n)

rep = s \* 5 # 'HelloHelloHelloHelloHello'

Métodos de las cadenas

Las cadenas en Python tienen métodos que realizan diversas operaciones con este tipo de datos.

Ejemplo: sacar (strip) los espacios en blanco sobrantes al inicio o al final de una cadena.

s = ' Hello '

t = s.strip() # 'Hello'

Ejemplo: Conversión entre mayúsculas y minúsculas.

s = 'Hello'

l = s.lower() # 'hello'

u = s.upper() # 'HELLO'

Ejemplo: Reemplazo de texto.

s = 'Hello world'

t = s.replace('Hello' , 'Hallo') # 'Hallo world'

Más métodos de cadenas:

Los strings (cadenas) ofrecen una amplia variedad de métodos para testear y manipular textos. Estos son algunos de los métodos:

s.endswith(suffix) # Verifica si termina con el sufijo

s.find(t) # Primera aparición de t en s (o -1 si no está)

s.index(t) # Primera aparición de t en s (error si no está)

s.isalpha() # Verifica si los caracteres son alfabéticos

s.isdigit() # Verifica si los caracteres son numéricos

s.islower() # Verifica si los caracteres son minúsculas

s.isupper() # Verifica si los caracteres son mayúsculas

s.join(slist) # Une una lista de cadenas usando s como delimitador

s.lower() # Convertir a minúsculas

s.replace(old,new) # Reemplaza texto

s.split([delim]) # Parte la cadena en subcadenas

s.startswith(prefix) # Verifica si comienza con un sufijo

s.strip() # Elimina espacios en blanco al inicio o al final

s.upper() # Convierte a mayúsculas

Mutabilidad de cadenas

Los strings son "inmutables" o de sólo lectura. Una vez creados, su valor no puede ser cambiado.

>>> s = 'Hello World'

>>> s[1] = 'a' # Intento cambiar la 'e' por una 'a'

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: 'str' object does not support item assignment

>>>

Esto implica que las operaciones y métodos que manipulan cadenas deben crear nuevas cadenas para almacenar su resultado.

Conversión de cadenas

Usá str() para convertir cualquier valor a cadena. El resultado es una cadena con el mismo contenido que hubiera mostrado el comando print() sobre la expresión entre paréntesis.

>>> x = 42

>>> str(x)

'42'

>>>

f-Strings

Las f-Strings son cadenas en las que ciertas expresiones son formateadas

>>> nombre = 'Naranja'

>>> cajones = 100

>>> precio = 91.1

>>> a = f'{nombre:>10s} {cajones:10d} {precio:10.2f}'

>>> a

' Naranja 100 91.10'

>>> b = f'Costo = ${cajones\*precio:0.2f}'

>>> b

'Costo = $9110.00'

>>>

Nota: Esto requiere Python 3.6 o uno más nuevo. El significado de los códigos lo veremos más adelante.

Ejercicios

En estos ejercicios vas a experimentar con operaciones sobre el tipo de dato string de Python. Hacelo en el intérprete interactivo para ver inmediatamente los resultados.

Recordamos:

En los ejercicios donde interactuás con el intérprete, el símbolo >>> es el que usa Python para indicarte que espera un nuevo comando. Algunos comandos ocupan más de una línea de código --para que funcionen, vas a tener que apretar 'enter' algunas veces. Acordate de no copiar el >>> de los ejemplos.

Comencemos definiendo una cadena que contiene una lista de frutas así::

>>> frutas = 'Manzana,Naranja,Mandarina,Banana,Kiwi'

>>>

Ejercicio 1.14: Extraer caracteres individuales y subcadenas

Los strings son vectores de caracteres. Tratá de extraer algunos carateres:

>>> frutas[0]

?

>>> frutas[1]

?

>>> frutas[2]

?

>>> frutas[-1] # Último caracter

?

>>> frutas[-2] # Índices negativos se cuentan desde el final

?

>>>

Como ya dijimos, en Python los strings son sólo de lectura. Verificá esto tratando de cambiar el primer caracter de frutas por una m minúscula 'm'.

>>> frutas[0] = 'm'

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: 'str' object does not support item assignment

>>>

Ejercicio 1.15: Concatenación de cadenas

A pesar de ser sólo de lectura, siempre podés reasignar una variable a una cadena nueva. Probá el siguiente comando que concatena la palabra "Pera" al final de frutas:

>>> frutas = frutas + 'Pera'

>>> frutas

'Manzana,Naranja,Mandarina,Banana,KiwiPera'

>>>

Ups! No es exactamento lo que queríamos. Reparalo para que quede 'Manzana,Naranja,Mandarina,Banana,Kiwi,Pera'.

>>> frutas = ?

>>> frutas

'Manzana,Naranja,Mandarina,Banana,Kiwi,Pera'

>>>

Agregá 'Melón'` al principio de la cadena:

>>> frutas = ?

>>> frutas

'Melón,Manzana,Naranja,Mandarina,Banana,Kiwi,Pera'

>>>

Podría parecer en estos ejemplos que la cadena original está siendo modificada, contradiciendo la regla de que las cadenas son de sólo lectura. No es así. Las operaciones sobre cadenas crean una nueva cadena cada vez. Cuando la variable frutas es reasignada, apunta a la cadena recientemente creada. Luego, la cadena vieja es destruida dado que ya no está siendo usada.

Ejercicio 1.16: Testeo de pertenencia (test de subcadena)

Experimentá con el operador in para buscar subcadenas. En el intérprete interactivo probá estas operaciones:

>>> 'Naranja' in frutas

?

>>> 'nana' in frutas

True

>>> 'Lima' in frutas

?

>>>

¿Por qué la verificación de 'nana' dió True?

Ejercicio 1.17: Iteración sobre cadenas

Usá el comando for para iterar sobre los caracteres de una cadena.

>>> cadena = "Ejemplo con for"

>>> for c in cadena:

print('caracter:', c)

# Mirá el output.

Modificá el código anterior de manera que dentro del ciclo el programa cuente cuántas letras "o" hay en la cadena.

Sugerencia: usá un contador como con los meses de la hipoteca.

Ejercicio 1.18: Geringoso rústico

Usá una iteración sobre el string cadena para agregar la sílaba 'pa', 'pe', 'pi', 'po', o 'pu' según corresponda luego de cada vocal.

>>> cadena = 'Geringoso'

>>> capadepenapa = ''

>>> for c in cadena:

?

>>> capadepenapa

Geperipingoposopo

Podés probar tu código cambiando la cadena inicial por otra palabra, como 'apa' o 'boligoma'.

Guardá el código en un archivo geringoso.py.

Ejercicio 1.19: Métodos de cadenas

En el intérprete interactivo experimentá con algunos de los métodos de cadenas introducidos antes.

>>> frutas.lower()

?

>>> frutas

?

>>>

Recordá, las cadenas son siempre de sólo lectura. Si querés guardar el resultado de una operación, vas a necesitás asignárselo a una variable:

>>> lowersyms = frutas.lower()

>>>

Probá algunas más:

>>> frutas.find('Mandarina')

?

>>> frutas[13:17]

?

>>> frutas = frutas.replace('Kiwi','Melón')

>>> frutas

?

>>> nombre = ' Naranja \n'

>>> nombre = nombre.strip() # Remove surrounding whitespace

>>> nombre

?

>>>

Ejercicio 1.20: f-strings

A veces querés crear una cadena que incorpore los valores de otras variables en ella.

Para hacer eso, usá una f-string. Por ejemplo:

>>> nombre = 'Naranja'

>>> cajones = 100

>>> precio = 91.1

>>> f'{cajones} cajones de {nombre} a ${precio:0.2f}'

'100 cajones de Naranja a $91.10'

>>>

Modificá el programa hipoteca.py del Ejercicio 1.11 de la sección anterior para que escriba su salida usando f-strings. Tratá de hacer que la salida quede bien alineada.

Ejercicio 1.21: Expresiones regulares

Una limitación de las operaciones básicas de cadenas es que no ofrecen ningún tipo de transformación usando patrones más sofisticados. Para eso vas a tener que usar el módulo re de Python y aprender a usar expresiones regulares. El manejo de estas expresiones es un tema en sí mismo. A continuación presentamos un corto ejemplo:

>>> texto = 'Hoy es 6/8/2020. Mañana será 7/8/2020.'

>>> # Encontrar las apariciones de una fecha en el texto

>>> import re

>>> re.findall(r'\d+/\d+/\d+', texto)

['6/8/2020', '7/8/2020']

>>> # Reemplazá esas apariciones, cambiando el formato

>>> re.sub(r'(\d+)/(\d+)/(\d+)', r'\3-\2-\1', texto)

'Hoy es 2020-8-6. Mañana será 2020-8-7.'

>>>

Para más información sobre el módulo re, mirá la documentación oficial en inglés o algún tutorial en castellano. Es un tema que escapa al contenido del curso pero te recomendamos que mires en detalle en algún momento. Aunque no justo ahora. Sigamos...

Comentario

A medida que empezás a usar Python es usual que quieras saber qué otras operaciones admiten los objetos con los que estás trabajando. Por ejemplo. ¿cómo podés averiguar qué operaciones se pueden hacer con una cadena?

Dependiendo de tu entorno de Python, podrás ver una lista de métodos disponibles apretando la tecla tab. Por ejemplo, intentá esto:

>>> s = 'hello world'

>>> s.<tecla tab>

>>>

Si al presionar tab no pasa nada, podés volver al viejo uso de la función dir(). Por ejemplo:

>>> s = 'hello'

>>> dir(s)

['\_\_add\_\_', '\_\_class\_\_', '\_\_contains\_\_', ..., 'find', 'format',

'index', 'isalnum', 'isalpha', 'isdigit', 'islower', 'isspace',

'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'partition',

'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit',

'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase',

'title', 'translate', 'upper', 'zfill']

>>>

dir() produce una lista con todas las operaciones que pueden aparecer luego del parámetro que le pasaste, en este caso s. También podés usar el comando help() para obtener más información sobre una operación específica:

>>> help(s.upper)

Help on built-in function upper:

upper(...)

S.upper() -> string

Return a copy of the string S converted to uppercase.

>>>

1.5 Listas

En esta sección estudiaremos listas que es el tipo de datos primitivo de Python para guardar colecciones ordenadas de valores.

Creación de Listas

Usá corchetes para definir una lista:

nombres = [ 'Rosita', 'Manuel', 'Luciana' ]

nums = [ 39, 38, 42, 65, 111]

A veces las listas son creadas con otros métodos. Por ejemplo, los elementos de una cadena pueden ser separados en una lista usando el método split():

>>> line = 'Pera,100,490.10'

>>> row = line.split(',') #la coma indica el elemento que separa

>>> row

['Pera', '100', '490.10']

>>>

Operaciones con listas

Las listas pueden almacenar elementos de cualquier tipo. Podés agregar nuevos elementos usando append():

nombres.append('Mauro') # Lo agrega al final

Usá el símbolo de adición + para concatenar listas:

s = [1, 2, 3]

t = ['a', 'b']

s + t # [1, 2, 3, 'a', 'b']

Las listas se indexan con números enteros, comenzando en 0.

nombres = [ 'Rosita', 'Manuel', 'Luciana' ]

nombres[0] # 'Rosita'

nombres[1] # 'Manuel'

nombres[2] # 'Luciana'

Los índices negativos cuentan desde el final.

nombres[-1] # 'Luciana'

Podés cambiar cualquier elemento de una lista.

nombres[1] = 'Juan Manuel'

nombres # [ 'Rosita', 'Juan Manuel', 'Luciana' ]

Y podés insertar elementos en una posición. Acordate que los índices comienzan a contar desde el 0.

nombres.insert(2, 'Iratxe') # Lo inserta en la posición 2.

nombres.insert(0, 'Iratxe') # Lo inserta como primer elemento.

La función len permite obtener la longitud de una lista.

nombres = ['Rosita','Manuel','Luciana']

len(nombres) # 3

Test de pertenencia a la lista (in, not in).

'Rosita' in nombres # True

'Diego' not in nombres # True

Se puede replicar una lista (s \* n).

s = [1, 2, 3]

s \* 3 # [1, 2, 3, 1, 2, 3, 1, 2, 3]

Iteradores de listas y búsqueda

Usá el comando for para iterar sobre los elementos de una lista.

for nombre in nombres:

# usá nombre

# e.g. print(nombre)

...

Para encontrar rápidamente la posición de un elemento en una lista, usá index().

nombres = ['Rosita','Manuel','Luciana']

nombres.index('Luciana') # 2

Si el elemento está presente en más de una posición, index() te va a devolver el índice de la primera aparición. Si el elemento no está en la lista se va a generar una excepción de tipo ValueError.

Borrar elementos

Podés borrar elementos de una lista tanto usando el valor del elemento como su posición:

# Usando el valor

nombres.remove('Luciana')

# Usando la posición

del nombres[1]

Al borrar un elemento no se genera un hueco. Los siguientes elementos se moverán para llenar el vacío. Si hubiera más de una aparición de un valor, remove() sólo sacará la primera aparición.

Ordenar una lista

Las listas pueden ser ordenadas "in-place", es decir, sin usar nuevas variables.

s = [10, 1, 7, 3]

s.sort() # [1, 3, 7, 10]

# Orden inverso

s = [10, 1, 7, 3]

s.sort(reverse=True) # [10, 7, 3, 1]

# Funciona con cualquier tipo de datos que tengan orden

s = ['foo', 'bar', 'spam']

s.sort() # ['bar', 'foo', 'spam']

Usá sorted() si querés generar una nueva lista ordenada en lugar de ordenar la misma:

t = sorted(s) # s queda igual, t guarda los valores ordenados

Listas y matemática

Cuidado: Las listas no fueron diseñadas para realizar operaciones matemáticas.

>>> nums = [1, 2, 3, 4, 5]

>>> nums \* 2

[1, 2, 3, 4, 5, 1, 2, 3, 4, 5]

>>> nums + [10, 11, 12, 13, 14]

[1, 2, 3, 4, 5, 10, 11, 12, 13, 14]

Específicamente, las listas no representan vectores ni matrices como en MATLAB, Octave, R, etc. Sin embargo, hay paquetes de Python que hacen muy bien ese trabajo (por ejemplo numpy).

Ejercicios

En este ejercicio, vamos a experimentar con el tipo de dato lista de Python. En la última sección, trabajaste con cadenas que hacían referencia a cajones de frutas.

>>> frutas = 'Frambuesa,Manzana,Naranja,Mandarina,Banana,Sandía,Pera'

Armá una lista con los nombres de frutas usando el comando split():

>>> lista\_frutas = frutas.split(',')

Ejercicio 1.22: Extracción y reasignación de elementos.

Probá un par de estos comandos para extraer un elemento:

>>> lista\_frutas[0]

'Frambuesa'

>>> lista\_frutas[1]

'Manzana'

>>> lista\_frutas[-1]

'Pera'

>>> lista\_frutas[-2]

'Sandía'

>>>

Intentá reasignar un valor:

>>> lista\_frutas[2] = 'Granada'

>>> lista\_frutas

['Frambuesa', 'Manzana', 'Granada', 'Mandarina', 'Banana', 'Sandía', 'Pera']

>>>

Hacé unas rebanadas (slices) de la lista:

>>> lista\_frutas[0:3]

['Frambuesa', 'Manzana', 'Granada']

>>> lista\_frutas[-2:]

['Sandía', 'Pera']

>>>

Creá una lista vacía y agregale un elemento.

>>> compra = []

>>> compra.append('Pera')

>>> compra

['Pera']

Podés incluso reasignar una lista a una porción de otra lista. Por ejemplo:

>>> lista\_frutas[-2:] = compra

>>> lista\_frutas

['Frambuesa', 'Manzana', 'Granada', 'Mandarina', 'Banana', 'Pera']

>>>

Cuando hacés esto, la lista del lado izquierdo (lista\_frutas) va a cambiar su tamaño para que encaje la lista del lado derecho (compra). En el ejemplo de arriba los últimos dos elementos de la lista\_frutas fueron reemplazados por un solo elemento en la lista compra.

Ejercicio 1.23: Ciclos sobre listas

El ciclo for funciona iterando sobre datos en una secuencia. Antes vimos que podíamos iterar sobre los caracteres de una cadena (las cadenas son secuencias). Ahora veremos que podemos iterar sobre listas también. Verificá esto tipeando lo que sigue y viendo qué pasa:

>>> for s in lista\_frutas:

print('s =', s)

Ejercicio 1.24: Test de pertenencia

Usá los operadores in o not in para verificar si 'Granada','Lima', y 'Limon' pertenecen a la lista de frutas.

>>> # ¿Está 'Granada' IN `lista\_frutas`?

True

>>> # ¿Está 'Lima' IN `lista\_frutas`?

False

>>> # ¿Está 'Limon' NOT IN `lista\_frutas`?

True

>>>

Ejercicio 1.25: Adjuntar, insertar y borrar elementos

Usá el método append() para agregar 'Mango' al final de lista\_frutas.

>>> # agregar 'Mango'

>>> lista\_frutas

['Frambuesa', 'Manzana', 'Granada', 'Mandarina', 'Banana', 'Pera', 'Mango']

>>>

Usá el método insert() para agregar 'Lima' como segundo elemento de la lista.

>>> # Insertar 'Lima' como segundo elemento

>>> lista\_frutas

['Frambuesa', 'Lima', 'Manzana', 'Granada', 'Mandarina', 'Banana', 'Pera', 'Mango']

>>>

Usá el método remove() para borrar 'Mandarina' de la lista.

>>> # Borrar 'Mandarina'

>>> lista\_frutas

['Frambuesa', 'Lima', 'Manzana', 'Granada', 'Banana', 'Pera', 'Mango']

>>>

Agregá una segunda copia de 'Banana' al final de la lista.

Observación: es perfectamente válido tener valores duplicados en una lista.

>>> # Agregar 'Banana'

>>> lista\_frutas

['Frambuesa', 'Lima', 'Manzana', 'Granada', 'Banana', 'Pera', 'Mango', 'Banana']

>>>

Usá el método index() para determinar la posición de la primera aparición de 'Banana' en la lista.

>>> # Encontrar la primera aparición de 'Banana'

>>> lista\_frutas

4

>>> lista\_frutas[4]

'Banana'

>>>

Contá la cantidad de apariciones de 'Banana' en la lista:

>>> lista\_frutas.count('Banana')

2

>>>

Borrá la primera aparición de 'Banana'.

>>> # Borrar la primer aparición de 'Banana'

>>> lista\_frutas

['Frambuesa', 'Lima', 'Manzana', 'Granada', 'Pera', 'Mango', 'Banana']

>>>

Para que sepas, no hay un método que permita encontrar o borrar todas las apariciones de un elemento en un a lista. Más adelante veremos una forma elegante de hacerlo.

Ejercicio 1.26: Sorting

¿Querés ordenar una lista? Usá el método sort(). Probalo:

>>> lista\_frutas.sort()

>>> lista\_frutas

['Banana', 'Frambuesa', 'Granada', 'Lima', 'Mango', 'Manzana', 'Pera']

>>>

¿Y si ordenamos al revés?

>>> lista\_frutas.sort(reverse=True)

>>> lista\_frutas

['Pera', 'Manzana', 'Mango', 'Lima', 'Granada', 'Frambuesa', 'Banana']

>>>

Observación: acordate de que el método sort() modifica el contenido de la misma lista in-place. Los elementos son reordenados moviéndolos de una posición a otra, pero no se crea una nueva lista.

Ejercicio 1.27: Juntar múltiples cadenas

Si querés juntar las cadenas en una lista, usá el método join() de los strings como sigue (ojo: parece un poco raro al principio).

>>> lista\_frutas = ['Banana', 'Mango', 'Frambuesa', 'Pera', 'Granada', 'Manzana', 'Lima']

>>> a = ','.join(lista\_frutas)

>>> a

'Banana,Mango,Frambuesa,Pera,Granada,Manzana,Lima'

>>> b = ':'.join(lista\_frutas)

>>> b

'Banana:Mango:Frambuesa:Pera:Granada:Manzana:Lima'

>>> c = ''.join(lista\_frutas)

>>> c

'BananaMangoFrambuesaPeraGranadaManzanaLima'

>>>

Ejercicio 1.28: Listas de cualquier cosa

Las listas pueden contener cualquier tipo de objeto, incluyendo otras listas (serían 'listas anidadas').

Probá esto:

>>> nums = [101, 102, 103]

>>> items = ['spam', lista\_frutas, nums]

>>> items

['spam', ['Banana', 'Mango', 'Frambuesa', 'Pera', 'Granada', 'Manzana', 'Lima'], [101, 102, 103]]

Fijate bien el output. items es una lista con tres elementos. El primero es un string, pero los otros dos elementos son listas.

Podés acceder a los elementos de las listas anidadas usando múltiples operaciones de acceso por índice.

>>> items[0]

'spam'

>>> items[0][0]

's'

>>> items[1]

['Banana', 'Mango', 'Frambuesa', 'Pera', 'Granada', 'Manzana', 'Lima']

>>> items[1][1]

'Mango'

>>> items[1][1][2]

'n'

>>> items[2]

[101, 102, 103]

>>> items[2][1]

102

>>>

A pesar de que es técnicamente posible hacer una estructura de listas muy complicada, como regla general, es mejor mantener las cosas simples. Lo más usual es guardar en las listas muchos elementos del mismo tipo. Por ejemplo, una lista sólo de números o una lista de cadenas. Mezclar diferentes tipos de datos en una misma lista puede volverse conceptualmente difuso, así que mejor lo evitamos.

Ejercicio 1.29: Traductor (rústico) al lenguaje inclusivo

Queremos hacer un traductor que cambie las palabras masculinas de una frase por su versión neutra. Como primera aproximación, completá el siguiente código para reemplazar todas las letras 'o' que figuren en el último o anteúltimo caracter de cada palabra por una 'e'. Por ejemplo 'todos somos programadores' pasaría a ser 'todes somes programadores'. Guardá tu código en el archivo inclusive.py

>>> frase = 'todos somos programadores'

>>> palabras = frase.split()

>>> for palabra in palabras:

if ?

...

frase\_t = ?

print(frase\_t)

'todes somes programadores'

>>>

Probá tu código con 'Los hermanos sean unidos porque ésa es la ley primera', '¿cómo transmitir a los otros el infinito Aleph?' y 'Todos, tu también'. ¿Qué falla en esta última? (¡no hace falta que lo resuelvas!)

1.6 Cierre de la clase

En esta clase aprendimos a correr el intérprete de Python desde la línea de comandos para usarlo como una calculadora. Aprendimos a editar programas con un editor de texto y a correrlos en la terminal. Vimos diferentes tipos de datos: números enteros, números de punto flotante, cadenas y listas.

Al enviar tus archivos entendemos que leíste y estás de acuerdo con el código de honor del curso en el que hablamos de las reglas que rigen en este curso para evitar el plagio así como otros aspectos importantes sobre qué se puede compartir y qué no. En caso contrario no envíes tus archivos y contactate con les docentes.

Para cerrar esta clase te pedimos dos cosas:

Que recopiles las soluciones de los siguientes ejercicios:

Ejercicio 1.5 Pelota (rebotes.py)

Ejercicio 1.11 Bonus de Hipoteca (hipoteca.py)

Ejercicio 1.13 Volumen de la esfera (esfera.py)

Ejercicio 1.18 Geringoso (geringoso.py)

Que completes este formulario usando como identificación tu dirección de mail. Al terminar vas a obtener un link para enviarnos tus ejercicios y te vamos a preguntar si querés participar de la revisión de ejercicios por pares (peer-review). En ese caso te vamos a mandar para que corrijas un par de ejercicios y vas a a recibir un par de correcciones.

Por favor, usá siempre la misma dirección de mail con la que te inscribiste al curso así podemos llevar registro de tus entregas.

Observación: Si el enunciado de un ejercicio te pide que lo corras con un input particular, por favor poné la salida que obtuviste como comentario en tu código.

Por último te recordamos que es fundamental para mantenerte al tanto de las novedades del curso así como para hacer consultas a los docentes que te sumes al grupo de Slack de la materia.