

Laboratorio de Datos

Verano 2024

Introducción a Python - parte 1

Contenido

- + Python
- + Tipos de datos básicos - operaciones
- + Tipos de datos que contienen otros datos.
- + Condicional y ciclos
- + Correr archivos .py
- + Funciones

Python

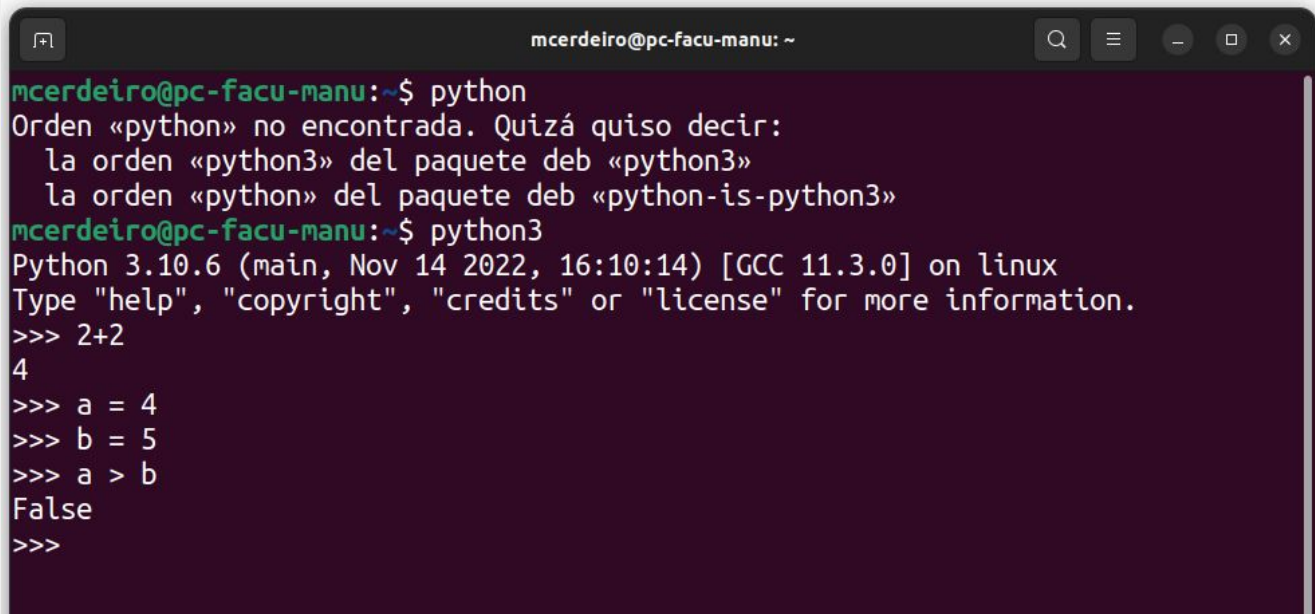


Instalación Python

- + código abierto (open source)
- + multiplataforma (sirve en linux, ios, windows...)
- + tiene muchas herramientas para ciencias de datos (y para muchas otras disciplinas)
- + alto nivel, sencillo de aprender
- + mucha mucha gente lo usa - facilita la consulta

Python desde la terminal

En una terminal, usamos "python" o "python3" para abrir un intérprete de python.

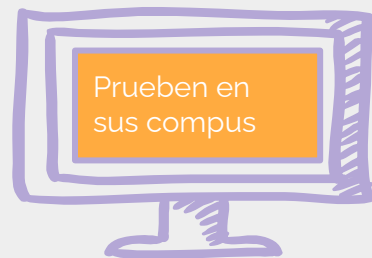


```
mcerdeiro@pc-facu-manu: ~  
mcerdeiro@pc-facu-manu:~$ python  
Orden «python» no encontrada. Quizá quiso decir:  
  la orden «python3» del paquete deb «python3»  
  la orden «python» del paquete deb «python-is-python3»  
mcerdeiro@pc-facu-manu:~$ python3  
Python 3.10.6 (main, Nov 14 2022, 16:10:14) [GCC 11.3.0] on linux  
Type "help", "copyright", "credits" or "license" for more information.  
>>> 2+2  
4  
>>> a = 4  
>>> b = 5  
>>> a > b  
False  
>>>
```

Python desde la terminal

Al terminar, salimos con `exit()`

```
mcerdeiro@pc-facu-manu: ~  
mcerdeiro@pc-facu-manu:~$ python  
Orden «python» no encontrada. Quizá quiso decir:  
  la orden «python3» del paquete deb «python3»  
  la orden «python» del paquete deb «python-is-python3»  
mcerdeiro@pc-facu-manu:~$ python3  
Python 3.10.6 (main, Nov 14 2022, 16:10:14) [GCC 11.3.0] on linux  
Type "help", "copyright", "credits" or "license" for more information.  
>>> 2+2  
4  
>>> a = 4  
>>> b = 5  
>>> a > b  
False  
>>> exit()  
mcerdeiro@pc-facu-manu:~$
```



Asignación de variables

```
a = 3
```

Esta instrucción crea una variable llamada **a** y le asigna el valor 3.

Si ya había una variable **a**, la pisa. Ahora la variable **a** vale 3.

```
a = 5 # ahora a vale 5
```

```
# el contenido posterior a un símbolo numeral (#)
```

```
# es un comentario
```

```
# python no lo ejecuta
```

Python Tutor - <https://pythontutor.com/>

Python Tutor: Visualize code in Python, JavaScript, C, C++, and Java

Python 3.6
[known limitations](#)

```
1 a = 3
→ 2 b = 10
→ 3 c = a + b
4 a = 5
```

[Edit this code](#)

→ line that just executed

→ next line to execute



<< First

< Prev

Next >

Last >>

Step 3 of 4

Frames

Objects

Global frame

a	3
b	10

Tipos de datos básicos

Tipos de datos básicos

- + int: representan números enteros, 1, 2, -5, 102978
- + float: representan números reales, 56.842
- + bool: representan valores booleanos, True/False (1/0).

Números enteros

280 + 16 # Este valor no se lo asigna a ninguna variable

base = 3

altura = 4

sup_rectangulo = base*altura

Calcula la superficie del rectángulo

20//3 # el símbolo // es para la división entera

15%4 # el símbolo % calcula el resto de la división entera

Variables float

```
base = 3
```

```
altura = 5
```

```
sup_triangulo = (base*altura)/2
```

Variables booleanas

Operaciones con bool

- + and
- + or
- + not

Representan la conjunción,
disyunción y negación,
respectivamente.

```
a = True
```

```
b = False
```

```
c = a and b
```

```
d = a or b
```

```
e = not a or b # a → b
```

Operaciones de int/float a bool

Al realizar comparaciones obtenemos valores de verdad (bool):

```
a = 8
```

```
b = 15
```

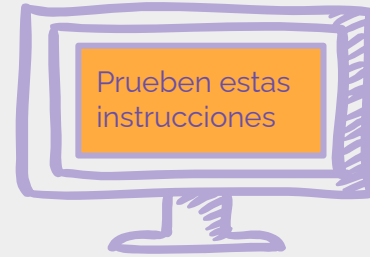
```
x = 2
```

```
a > b # False
```

```
(a > b) or (a > x) # True
```

Ejercicio. ¿Qué valor de verdad se obtiene?

```
(x < a) and (10*x >= b) and not (x == a - b)
```



Datos que contienen datos

- + Cadenas de caracteres
- + Listas
- + Tuplas
- + Conjuntos

Cadenas de caracteres - *Strings*

Son la forma de manipular texto.

Se construyen con comillas simples o dobles.

```
a = 'Hoy es martes'
```

```
b = 'empiezo labo de datos'
```

```
c = a + b # queda medio feo
```

```
c = a + ', ' + b # así va mejor
```


Cadenas de caracteres - *Strings*

Posiciones y porciones

```
a = 'Hoy es lunes'
```

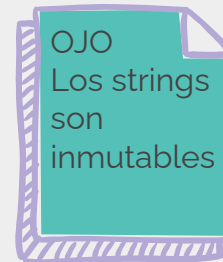
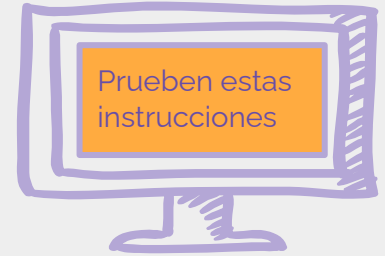
```
a[4] # lo que tiene a en la posición 4
```

```
a[4:8] # de 4 hasta pre-8
```

```
a[4:]
```

```
a[:5]
```

```
a[1] = 'x'
```



Cadenas de caracteres - *Strings*

La función **print** imprime en pantalla:

```
a = 'Hoy es lunes'
```

```
print(a)
```

```
print('ahora imprimo otra cadena')
```

```
x = 5
```

```
print(x) # también imprime cosas que no son strings
```

```
listita = [1,1,2,3,4,8]
```

```
print(listita)
```

Cadenas de caracteres - *Strings*

```
a = 'Hello' + 'World'    # concatenación: 'HelloWorld'
b = 'Say ' + a           # 'Say HelloWorld'
s = 'Hello'
len(s)                   # longitud (da 5)
'e' in s                 # test de pertenencia
'x' in s                 # False
'hi' not in s            # True
rep = s * 5              # 'HelloHelloHelloHelloHello'
l = s.lower()            # 'hello'
u = s.upper()            # 'HELLO'
```

Listas

Pueden contener todo tipo de variables.

Las listas se usan mucho. Se construyen con corchetes [].

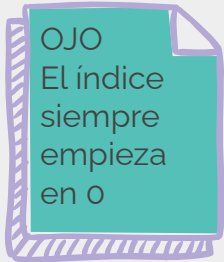
```
lista_num = [10, 43, 22, 5, 63, 101, -5, 3]
```

```
lista_nombres = ['Julia', 'Luciana', 'Manuel']
```

```
lista_num[0] # accedo al primer elemento de la lista
```

```
lista_nombres[1] # 'Luciana'
```

```
lista_vacia = []
```



OJO
El índice
siempre
empieza
en 0

Listas

Las listas se usan mucho. Se construyen con corchetes [].

```
lista_num = [10, 43, 22, 5, 63, 101, -5, 3]
```

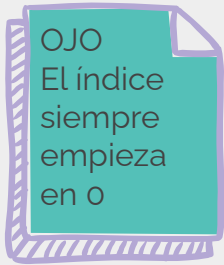
```
lista_num[0] # accedo al primer elemento de la lista
```

```
lista_num[3]
```

```
lista_num[7]
```

```
lista_num[1:5] # rebanada/slice
```

```
len(lista_num) # 8
```



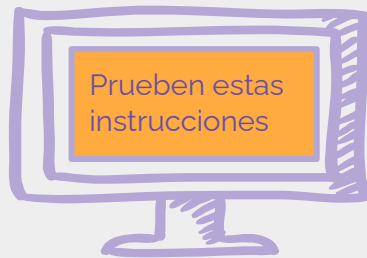
OJO
El índice
siempre
empieza
en 0

Listas

```
lista_num = [10, 43, 22, 5, 63, 101, -5, 3]
lista_nombres = ['Julia', 'Luciana', 'Manuel']
lista_num.append(28) # agrego un elemento al final
lista_nombres[1] = 'Lucía' # cambio el segundo elemento
22 in lista_num # me fijo si el elemento 22 está
'Manuel' in lista_nombres
25 not in lista_num # me fijo si NO está
nombres.remove('Julia')
lista_num.sort()
```

Listas

```
frutas = 'Frambuesa,Manzana,Naranja,Mandarina,Banana,Sandía,Pera'
lista_frutas = frutas.split(',') # separa en las comas
lista_frutas[0]    # Frambuesa
lista_frutas[1]
lista_frutas[-1]   # último elemento
lista_frutas[-2]   # Sandía
lista_frutas.append('Limón')
lista_frutas.insert(0, 'Limón') # insertar al principio
lista_frutas += ['Frutilla', 'Palta']
```



Listas

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

lista_frutas = frutas.split(',') # separa en las comas

compras = ['café', lista_frutas, 'huevos'] # tiene una lista adentro!

compras[1] ?

compras[1][1] ?

compras[1][1][1] ???

mandados = [compras, 'cajero', 'pasear al perro']

len(mandados) ?

lista_frutas in mandados # ??
```


compras

['café', lista_frutas, 'huevos']

compras[1]

[['café', ['Frambuesa', 'Manzana', 'Naranja'], 'huevos'], 'cajero',
'pasear al perro']

compras[1][1]

mandados

```
[compras, 'cajero', 'pasear al perro']
```

```
[['café', lista_frutas, 'huevos'], 'cajero', 'pasear al perro']
```

```
[['café', ['Frambuesa', 'Manzana', 'Naranja'], 'huevos'], 'cajero',  
'pasear al perro']
```

Matrices como listas de listas

Podemos utilizar listas de listas para definir matrices.
Cada lista representa una fila.

Ejemplos:

$M = [[1, 2], [3, 4]]$

1	2
3	4

$N = [[1, 0, 3], [0, 5, 5], [8, -1, 1], [10, 3, 1]]$

1	0	3
0	5	5
8	-1	1
10	3	1

Matrices como listas de listas

¿Cómo accedemos al elemento de la posición (i,j)?

`M[0] ?`

`M[0][1] ?`

1	2
3	4

`N[1][2] ?`

`N[?][?]` # quiero acceder al -1

1	0	3
0	5	5
8	-1	1
10	3	1

Tuplas

Son como vectores. Se arman con paréntesis ().

a = (2, 4)

b = (-1, 2)

c = a + b

d = (1, 5, 10, 3, 7)

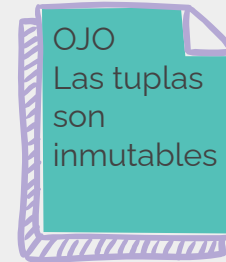
a[1]

d[9]

d[1:5]

d[2] = 11

d = (1, 5, 11, 3, 7)



Conjuntos

Se arman con llaves {}.

```
a = {2, 4}
```

```
citricos = {'Naranja', 'Limón', 'Mandarina'}
```

```
citricos = set(['Naranja', 'Limón', 'Mandarina'])
```

```
'Naranja' in citrics
```

```
citricos.add('Pomelo')
```

```
citricos.remove('Naranja')
```

```
len(a)
```

Ejercicio

Escribir un programa que decida si una palabra (de longitud 5) es palíndromo.

Ej. "NADAN" es palíndromo. "JUJUY" no lo es.

Cortamos 15 minutos

Condicional y ciclos

Condicional

Si queremos supeditar la instrucción a una determinada condición, utilizamos el condicional if.

```
if a > 0:
```

```
    print('a es un número positivo!')
```

Condicional

Si queremos supeditar la instrucción a una determinada condición, utilizamos el condicional if.

```
if a > 0:
```

2 puntitos

nueva línea

4 espacios

```
    print('a es un número positivo!')
```

nueva línea

Condicional

Si queremos supeditar la instrucción a una determinada condición, utilizamos el condicional if.

```
if a > 0:
```

2 puntitos

nueva línea

4 espacios

```
    print('a es un número positivo!')
```

nueva línea

```
if a == 0:
```

```
    print('a es cero...')
```

```
if b > a:
```

```
    print('b le ganó a a')
```

Condicional

Ejercicio. Completar este código.

```
a = 2
```

```
b = 6
```

```
c = 4
```

```
if - - -:
```

```
    print('b está entre a y c')
```

```
if - - -:
```

```
    print('b es mayor a c')
```

Condicional

Else y elif.

```
if condicion1:
    instruccion1
    instruccion2
elif condicion2:
    instruccion3
else:
    instruccion4
```

Ejemplo

```
if a % 4 == 0:
    b = a//4
elif a % 4 == 2:
    c = a//2
else:
    print('a es impar')
```

Condicional

case en cpp

Else y elif.

```
if condicion1:  
    instruccion1  
    instruccion2  
elif condicion2:  
    instruccion3  
else:  
    instruccion4
```

condiciones: bool



Ejemplo

```
if a % 4 == 0:  
    b = a//4  
elif a % 4 == 2:  
    c = a//2  
else:  
    print('a es impar')
```

Ciclo while

Para utilizar while vamos a considerar una condición que puede o no cumplirse (un bool True/False). Las instrucciones dentro del ciclo se ejecutarán mientras se satisface la condición.

```
i = 0
```

```
while i < 5:
```

```
    print(i)
```

```
    i += 1    # sumo de a 1
```

Ciclo while

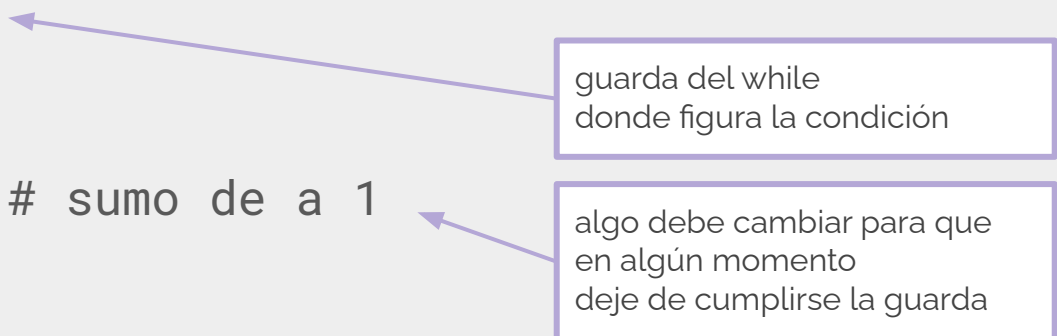
Para utilizar while vamos a considerar una condición que puede o no cumplirse (un bool True/False). Las instrucciones dentro del ciclo se ejecutarán mientras se satisface la condición.

```
i = 0
```

```
while i < 5:
```

```
    print(i)
```

```
    i += 1    # sumo de a 1
```



guarda del while
donde figura la condición

algo debe cambiar para que
en algún momento
deje de cumplirse la guarda

Ejercicio

Escribir un programa que imprima en pantalla los números enteros entre 0 y 213 que sean divisibles por 13.

Ejercicio

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?

Datos: espesor del billete: 0.11 mm, altura obelisco: 67.5 m.

(Está en la guía)

Ciclo for

Para utilizar el for vamos a considerar un iterador en la guarda. Las instrucciones dentro del ciclo for se ejecutarán en cada elemento generado por el iterador.

```
for i in range(5):  
    print(i)
```



guarda del for
donde figura el alcance

instrucciones a ejecutar en
cada paso

range()

```
for i in range(5):  
    print(i)
```

```
for i in range(0,5,1):  
    print(i)
```

```
for i in range(-5,-1):  
    print(i)
```

inicio, fin, paso

OJO
Fin significa
hasta antes
que ese
valor

valores por omisión:
inicio 0
paso 1

while vs. for

```
i = 0
```

```
while i < 5:
```

```
    print(i)
```

```
    i += 1
```

```
for i in range(0,5,1):
```

```
    print(i)
```

while vs. for

```
i = 0
```

```
while i < 5:
```

```
    print(i)
```

```
    i += 1
```

```
for i in range(0, 5, 1):
```

```
    print(i)
```

Ejercicio

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:
```

```
    COMPLETAR
```

```
print(capadepenapa)      # Geperipingoposopo
```

Luego hazelo con un while en vez del for.

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 21.6
4 12.96
5 7.776
6 4.6656
7 2.7994
8 1.6796
9 1.0078
10 0.6047
```

(Está en la guía)

Ejercicio

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 carácter de cada palabra por una 'e'. Por ejemplo 'todos somos programadores' pasaría a ser 'todes somes programadores'.

```
>>> frase = 'todos somos programadores'
>>> palabras = frase.split()
>>> for palabra in palabras:
    if ?
        ...
        frase_t = ?
        print(frase_t)
'todes somes programadores'
```

(Está en la guía)

Archivos .py

Uso de archivos .py

Podemos guardar el código como archivo de texto con extensión .py y luego ejecutarlo desde la terminal con python. Por ejemplo:

Creamos un archivo de texto, con este contenido, y lo guardamos como holamundo.py

```
# holamundo.py
```

```
a = "Hola, mundo!"  
b = a[:4]  
c = a[6:]  
print(a)
```

Uso de archivos .py

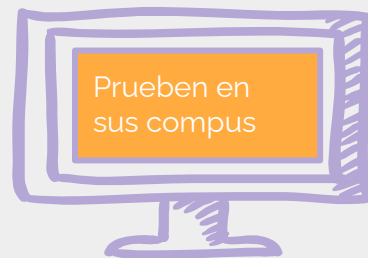
Ahora abrimos una terminal, nos situamos en el mismo directorio en el que está el archivo y ejecutamos "python3 holamundo.py"

```
mcerdeiro@clemen-i7:~/Downloads$ python3 holamundo.py  
Hola, mundo!
```

Uso de archivos .py

También podemos ejecutar con `-i` y quedarnos en el intérprete de python luego de la ejecución del programa. Las variables quedan en el estado final del programa.

```
mcerdeiro@clemen-i7:~/Downloads$ python3 holamundo.py
Hola, mundo!
mcerdeiro@clemen-i7:~/Downloads$ python3 -i holamundo.py
Hola, mundo!
>>> a
'Hola, mundo!'
>>> b
'Hola'
>>> c
'mundo!'
>>> b + c
'Holamundo!'
>>> exit()
mcerdeiro@clemen-i7:~/Downloads$
```



Funciones

Funciones

Las funciones son una herramienta para encapsular pedazos de código, facilitando su reutilización.

```
def sumar_le_uno(x):  
    res = x + 1  
    return res
```

```
def sumar(x,y):  
    res = x + y  
    return res
```

Funciones

Las funciones son una herramienta para encapsular pedazos de código, facilitando su reutilización.

```
def suma_gauss(n):  
    '''  
    Devuelve la suma de los primeros n enteros  
    '''  
    total = 0  
    while n > 0:  
        total += n  
        n -= 1  
    return total
```


Funciones

Definición de una función.

The diagram illustrates the syntax of a Python function definition with the following code and annotations:

```
def suma_gauss(n):  
    ...  
    Devuelve la suma de los primeros n enteros  
    ...  
    total = 0  
    while n > 0:  
        total += n  
        n -= 1  
    return total
```

Annotations and their corresponding code elements:

- nombre**: Points to the function name `suma_gauss`.
- parámetros / variables in**: Points to the parameter `n` in the function signature.
- 2 puntos**: Points to the colon `:` at the end of the function signature.
- 4 espacios**: Points to the indentation of the first line of the function body (`total = 0`).
- nueva línea**: Points to the line separator between the function signature and the first line of the body.
- out**: Points to the `return` statement.

Funciones

Definición de una función.

```
def suma_gauss(n):  
    '''  
    Devuelve la suma de los primeros  
    n enteros  
    '''  
    total = 0  
    while n > 0:  
        total += n  
        n -= 1  
    return total
```

Llamada (uso) de una función.

```
suma_gauss(10)  
  
x = 25  
suma_gaus(x)
```

Ejercicios

- ★ Definir una función **maximo(a,b)** que tome dos parámetros numéricos y devuelva el máximo.
- ★ Definir una función **tachar_pares(lista)** que tome una lista de números y devuelva una similar pero donde los números pares estén reemplazados por 'x'.

Ejercicio

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:

Ejercicio

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))
```

Ejercicio

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 posibilidad es inicializar mes en 0 y otra es inicializarla en 1. En el primer caso es probable 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

¿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
```

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

Ejercicios

- ★ Definir una función **dos_pertenece(lista)** que tome una lista y devuelva True si la lista tiene al 2 y False en caso contrario.
- ★ Definir una función **pertenece(lista, elem)** que tome una lista y un elemento y devuelva True si la lista tiene al elemento dado y False en caso contrario.
- ★ Definir una función **es_par(n)** que devuelva True si el número es par y False en caso contrario.
- ★ Definir una función **mas_larga(lista1, lista2)** que tome dos listas y devuelva la más larga.
- ★ Definir una función **cant_e** que tome una lista y devuelva la cantidad de letras 'e' que tiene.
- ★ Definir una función **sumar_unos** que tome una lista, les sume 1 a todos sus elementos, y devuelva la lista modificada.
- ★ Definir la función **mezclar(cadena1, cadena2)** que tome dos strings y devuelva el resultado de intercalar elemento a elemento. Por ejemplo: si intercalamos Pepe con Jose daría PJeopsee.