INTRODUCCIÓN A PYTHON

Universidad del Valle de Guatemala

python™

"Python is a powerful, easy-to-use, object-oriented programming language"

Tiene un alto desempeño, corre en todos los sistemas operativos y es extremadamente versátil.

VERSATILIDAD DE PYTHON

Scripts y programas	Desarrollo web	Ciencia de datos
Como una calculadora (desde la línea de comandos)	Python como lenguaje en el servidor	Conjunto muy rico de librerías
Scripts o programas complejos	Aplicaciones o REST API	Reunir, importar y limpiar datos
Aplicaciones de escritorio con una interfaz de usuario.	Entornos como Flask o Django	Análisis estadístico, gráficas, aprendizaje automático

python™

"Python is a powerful, easy-to-use, object-oriented programming language"

Tiene un alto desempeño, corre en todos los sistemas operativos y es extremadamente versátil.

python™

"Python is a powerful, easy-to-use, object-oriented programming language"

Tiene un alto desempeño, corre en todos los sistemas operativos y es extremadamente versátil.

Python tiene una sintaxis clara y simple, posee gran variedad de funciones incluidas y ofrece una buena documentación.

python™

"Python is a powerful, easy-to-use, object-oriented programming language"

Tiene un alto desempeño, corre en todos los sistemas operativos y es extremadamente versátil.

Python tiene una sintaxis clara y simple, posee gran variedad de funciones incluidas y ofrece una buena documentación.

Python incorpora objetos, clases, herencia y permite trabajar con estructuras de datos complejas.

Top Machine Learning Languages on GitHub

1. Python
2. C++
3. JavaScript
4. Java
5. C#
6. Julia
7. Shell
8. R
9. TypeScript
10. Scala

GitHub

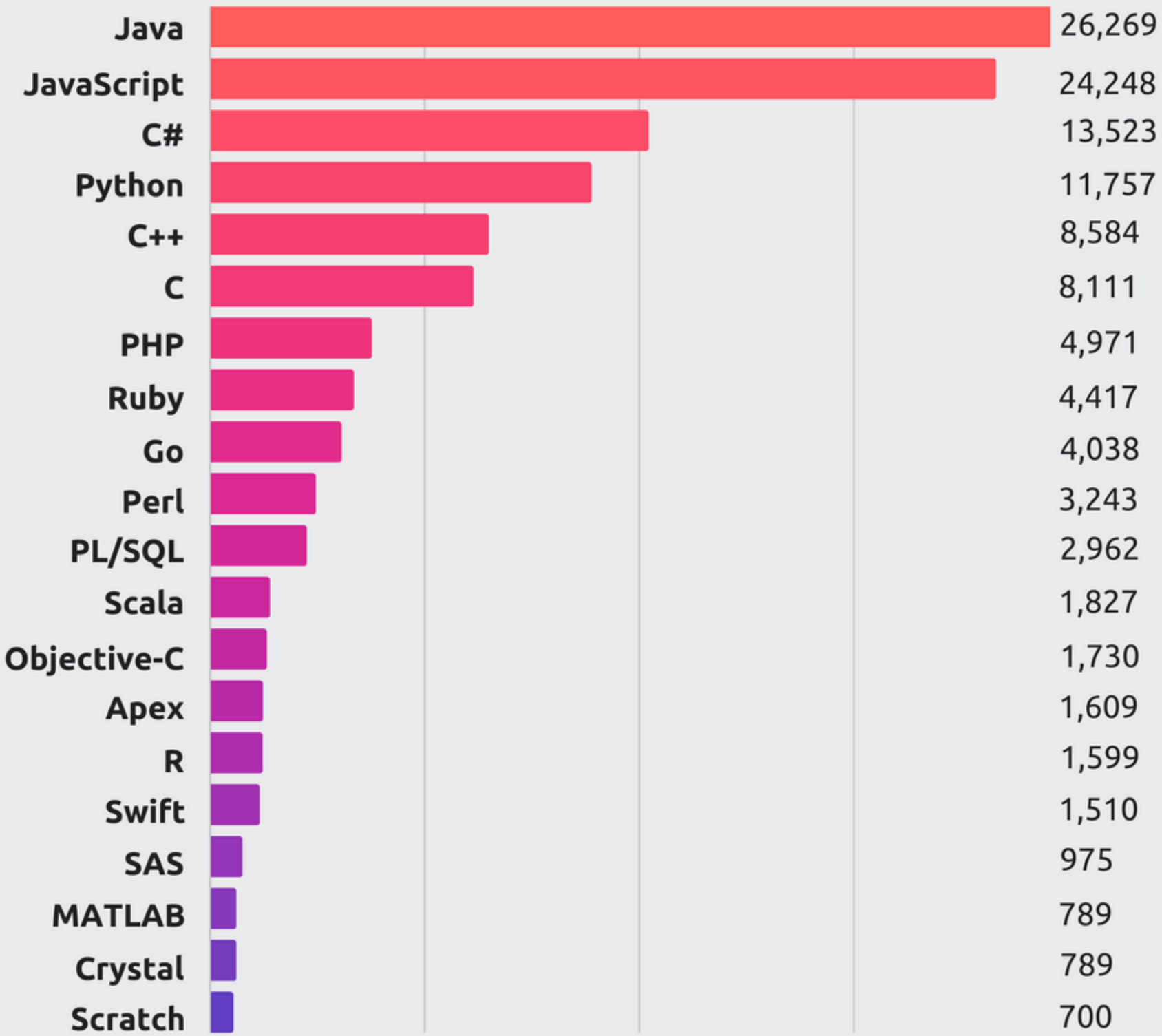
Packages Imported by Machine Learning Projects on GitHub

1	numpy	74%
2	scipy	47%
3	pandas	41%
4	matplotlib	40%
5	scikit-learn	38%
6	six	31%
7	tensorflow	24%
8	requests	23%
9	python-dateutil	22%
10	pytz	21%

GitHub

Most In-Demand Languages

Indeed Job Openings - Dec. 2017



INSTALACIÓN

Una de las maneras más limpias de instalar una distribución de Python es a través de ***Miniconda****. Se puede descargar en:*

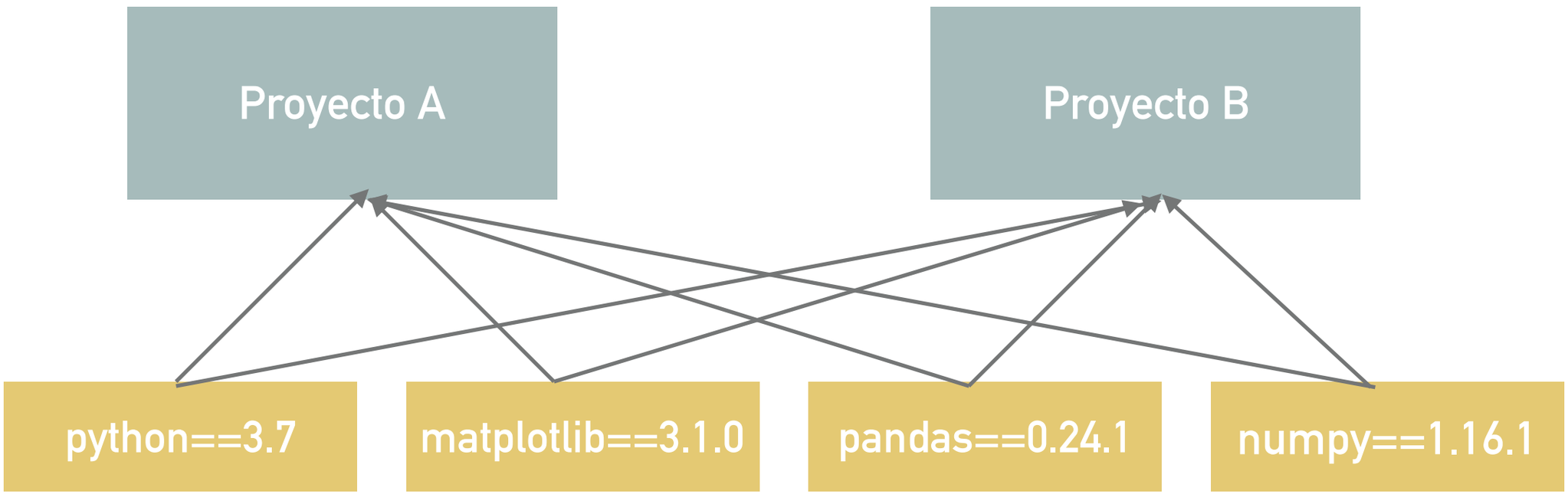
https://docs.conda.io/en/latest/miniconda.html

Instalamos la versión 3.7 de Python para el sistema operativo que usemos (Windows, Mac OS X o Linux).

Miniconda *contiene* ***conda****, que es un administrador de paquetes.*

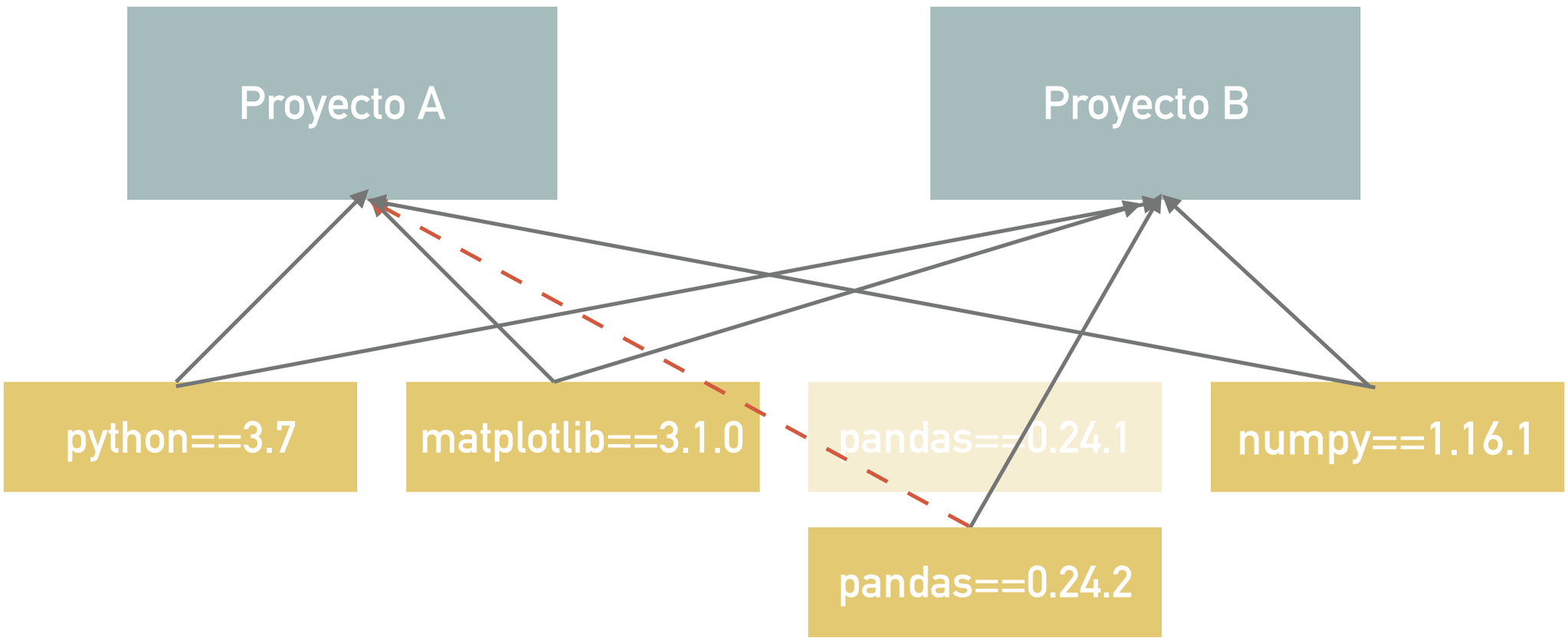
CREAR UN AMBIENTE (ENVIRONMENT)

Cada proyecto que realicemos va a depender de un conjunto de librerías externas, es decir, de librerías que no son parte de Python (una para hacer estadística, otra para graficar, una para leer bases de datos, etc.).



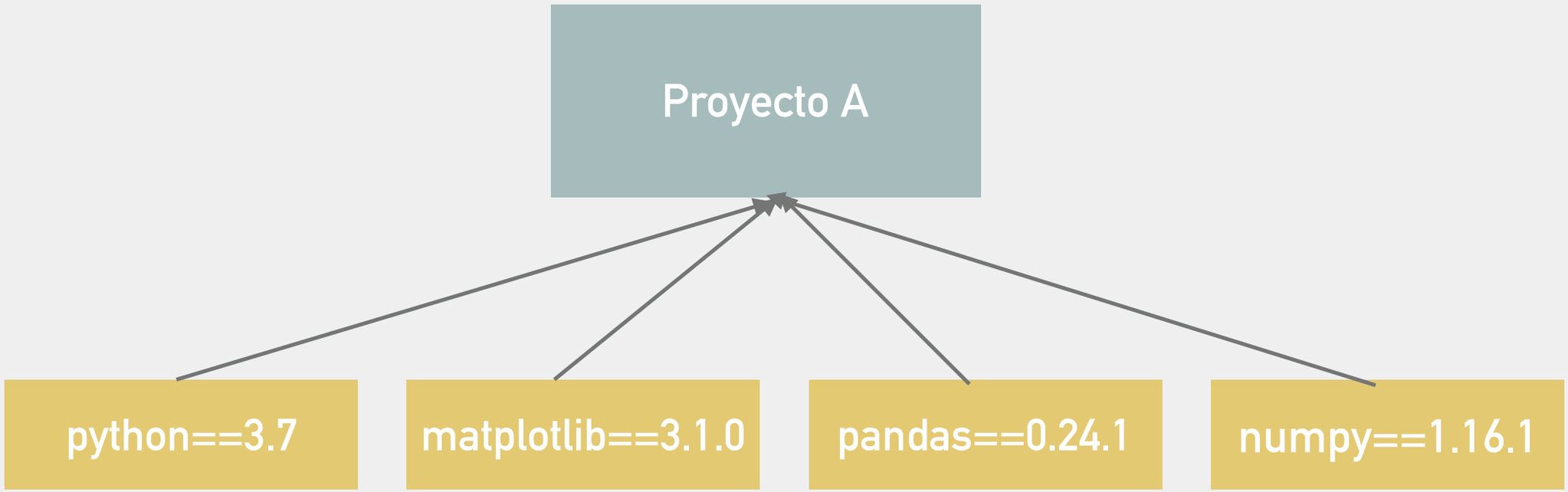
CREAR UN AMBIENTE (ENVIRONMENT)

Cada proyecto que realicemos va a depender de un conjunto de librerías externas, es decir, de librerías que no son parte de Python (una para hacer estadística, otra para graficar, una para leer bases de datos, etc.).

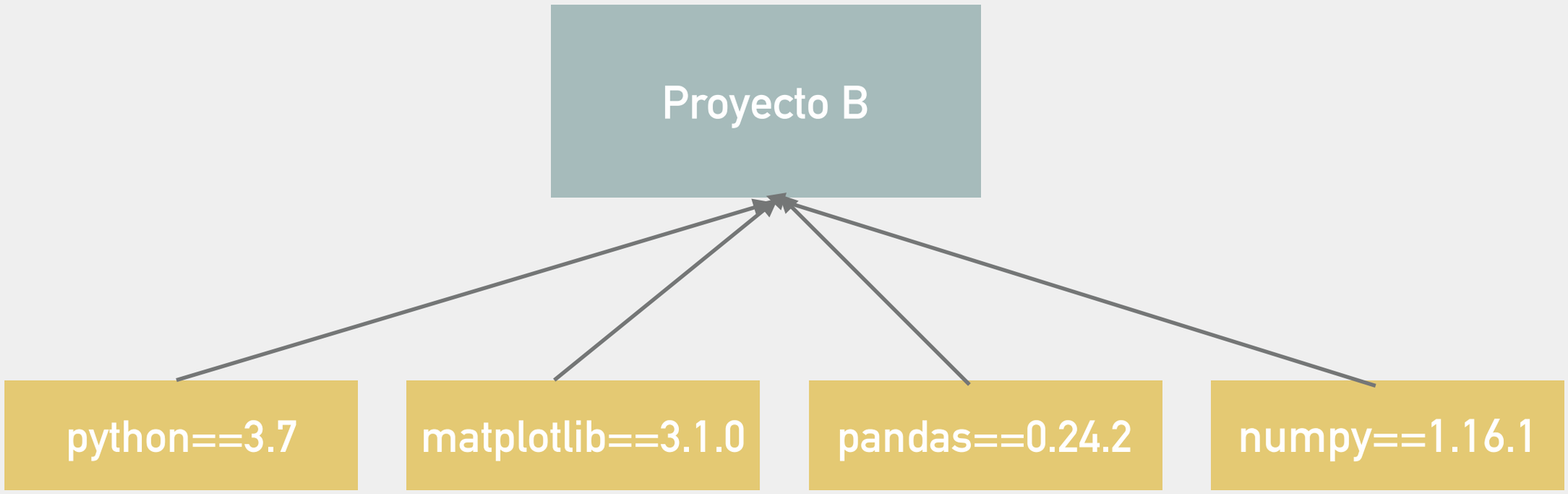


Si se genera alguna incompatibilidad, es necesario intervenir cada uno de los proyectos.

Ambiente A



Ambiente B



*Al instalar **miniconda**, por default se crea un ambiente llamado **base**.*

Crear un ambiente

```
conda create -n environment_a
conda create -n environment_b
```

Activar el ambiente A

```
conda activate environment_a
```

Instalar python o un paquete

```
conda install python
conda install python==3.7
conda install pandas==0.24.1 numpy
```

Actualizar un paquete o todos los paquetes

```
conda update pandas
conda update --all
```

Desinstalar un paquete

```
conda uninstall pandas
```

Desactivar el ambiente en que nos encontremos y volver a ***base***

```
conda deactivate
```

Mantener a ***conda*** *actualizado*

```
conda update conda
```

Eliminar un ambiente

```
conda deactivate
conda env remove -n environment_a
```

Más información sobre ***conda*** *(abreviada) en*

https://docs.conda.io/projects/conda/en/4.6.0/_downloads/52a95608c49671267e40c689e0bc00ca/conda-cheatsheet.pdf

INSTALAR UN EDITOR

*Un muy buen editor es **PyCharm***

https://www.jetbrains.com/pycharm/download

*Pueden descargar la edición **Community** o, con un correo institucional, como el de la UVG, pueden descargar la edición **Professional**. Info aquí:*

https://www.jetbrains.com/student/

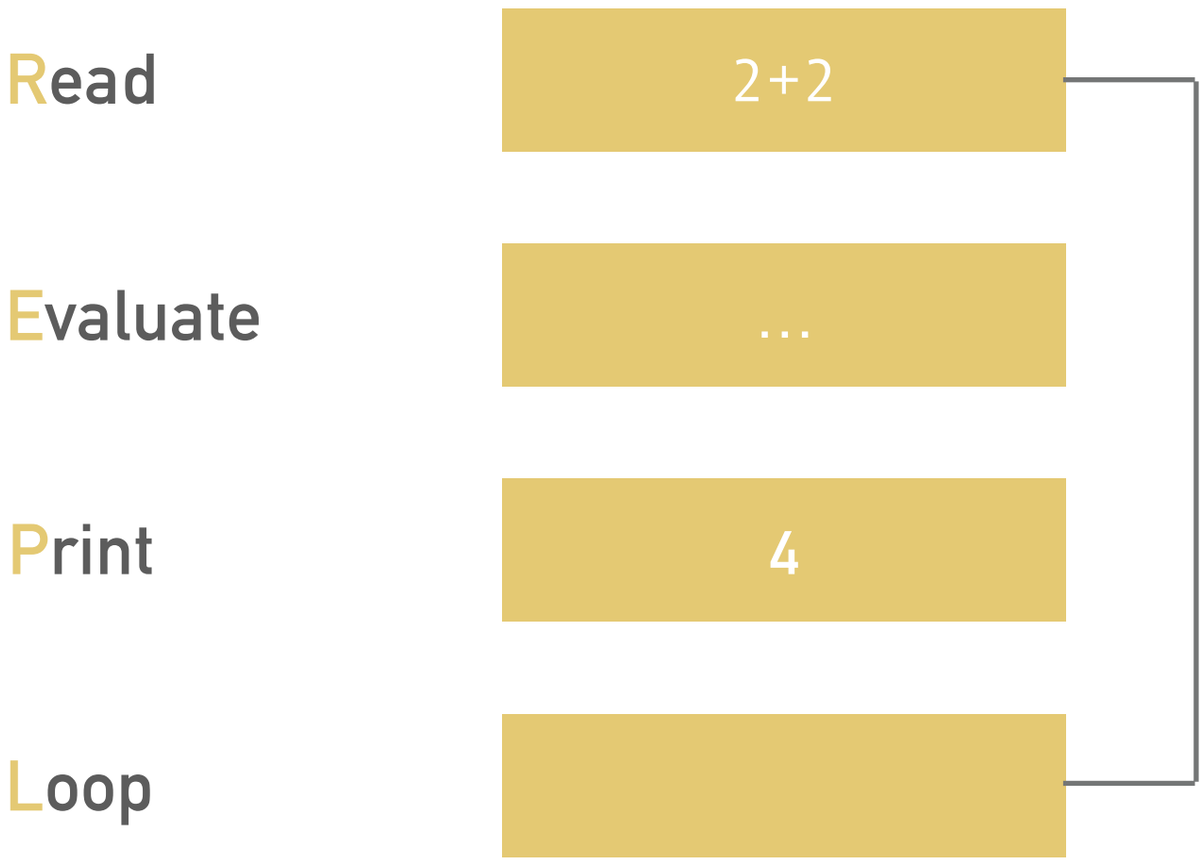
NUESTRO ENTORNO

```
conda create -n text_analytics
conda activate text_analytics
conda install -y python
```

Con esto, tenemos una instalación minimalista de Python (únicamente las librerías estándar).

USANDO EL REPL

```
python
```



JUPYTER-NOTEBOOK

```
conda install jupyter notebook
jupyter notebook
```

Ahora debe abrirse una ventana de navegador.

Los jupyter notebooks son cuadernos interactivos que ejecutan comandos en Python (o en muchos otros lenguajes, incluidos R).

TIPOS DE DATOS EN PYTHON

Números

Texto

Booleanos

Estructuras complejas

TIPOS DE DATOS EN PYTHON

Números

Enteros	`int`	`10`	`-3`
Flotantes	`float`	`1.8`	`-6.795`

Texto

Booleanos

Estructuras complejas

TIPOS DE DATOS EN PYTHON

Números	Enteros	`int`	`10`	`-3`
	Flotantes	`float`	`1.8`	`-6.795`
Texto	Texto	`str`	`'hola'`	`"mundo"`
Booleanos				
Estructuras complejas				

TIPOS DE DATOS EN PYTHON

Números	Enteros	`int`	`10`	`-3`
	Flotantes	`float`	`1.8`	`-6.795`
Texto	Texto	`str`	`'hola'`	`"mundo"`
Booleanos	Booleanos	`bool`	`True`	`False`
Estructuras complejas				

TIPOS DE DATOS EN PYTHON

Números	Enteros	`int`	`10`	`-3`
	Flotantes	`float`	`1.8`	`-6.795`
Texto	Texto	`str`	`'hola'`	`"mundo"`
Booleanos	Booleanos	`bool`	`True`	`False`
Estructuras complejas	Diccionarios, objectos, etc.			

VARIABLES

```
nombre = 'Pedro'
apellido = 'Aguilar'
edad = 34
altura = 1.79

nombre_completo = nombre + ' ' + apellido
nombre_completo = f'{nombre} {apellido}’
nombre_completo = '{} {}'.format(nombre, apellido)
```

Cuando sea mi cumpleaños

```
edad = edad + 1
```

2. El resultado se ***asigna*** *a la variable*

1. Python procesa el lado derecho

NÚMEROS

Enteros

`15, -55, 5421`

`int()` convierte otros tipos a enteros

Flotantes

`3.1416, 1.74, -2.718`

`float()` convierte otros tipos a flotantes

OPERACIONES

+	7+3	10
-	7-3	4
*	7*3	21
/	7/3	2.3333333333333
//	7//3	2
%	7%3	1
**	7**3	343

PRECISIÓN

```
1 - 0.9
1 - 0.5
1 - 0.75
```

La computadora procesa a los números decimales como sumas de potencias negativas de 2. Por ejemplo:

$$0.9 = 1/2 + 1/2^2 + 1/2^3 + 1/2^6 + 1/2^7 + 1/2^10 + 1/2^11 + \cdots$$

$$0.75 = 1/2 + 1/2^2$$

$$0.5 = 1/2$$

TEXTO

```
nombre = 'Mi nombre es Pedro'
frase = "El personaje se hace llamar ‘Chinaski’"
texto = """ La llamada Gran Recesión de la economía de
Estados Unidos, que arrancó oficialmente en diciembre de
2007, se ha consagrado ya como la recesión más grave desde
la Segunda Guerra Mundial . . .
"""
```

LISTAS

index 0 1 2 3

```
[ 'texto' , 12.9 , True , ['anidado', 8] ]
```

longitud: 4

```
lista[2] # == True
lista[1] # == 12.9
lista[1:3] # == [12.9, True]
lista[-1] # == [['anidado', 8]]
lista[:2] # == ['texto', 12.9]
lista[2:] # == [True, ['anidado', 8]]
```

MODIFICANDO LISTAS

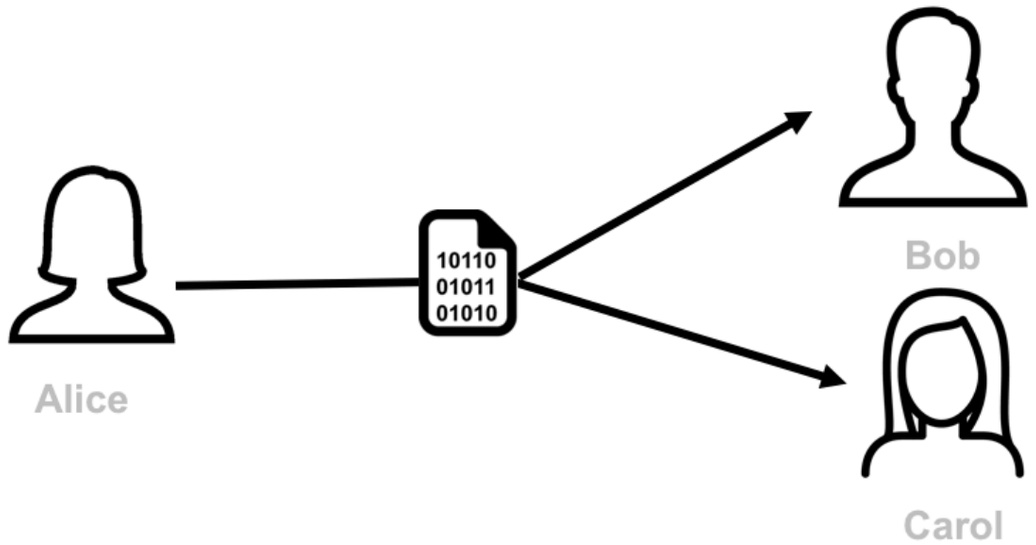
```
lista = [0,1,2,3,4]
lista.pop() # 4, lista == [0,1,2,3]
lista.append(4) # lista == [0,1,2,3,4]
lista[3] = 10 # lista == [0,1,2,10,4]
```

FUNCIONES

Definimos bloques de funcionalidad que pueden ser ejecutados modularmente.

```
def saludar():
    print('hola')

saludar()
saludar()
```

BLOCKCHAIN



Double-Spending Problem: If Alice sends money in digital format to Bob, Bob cannot know for sure if Alice has deleted her copy of the file and she can choose to send the same file to Carol.

Physical Cash

There is no double-spending problem with physical cash.

Centralized Digital Cash

Double-spending digital cash can be solve by a centralized 3rd party like a bank.

Decentralized Digital Cash

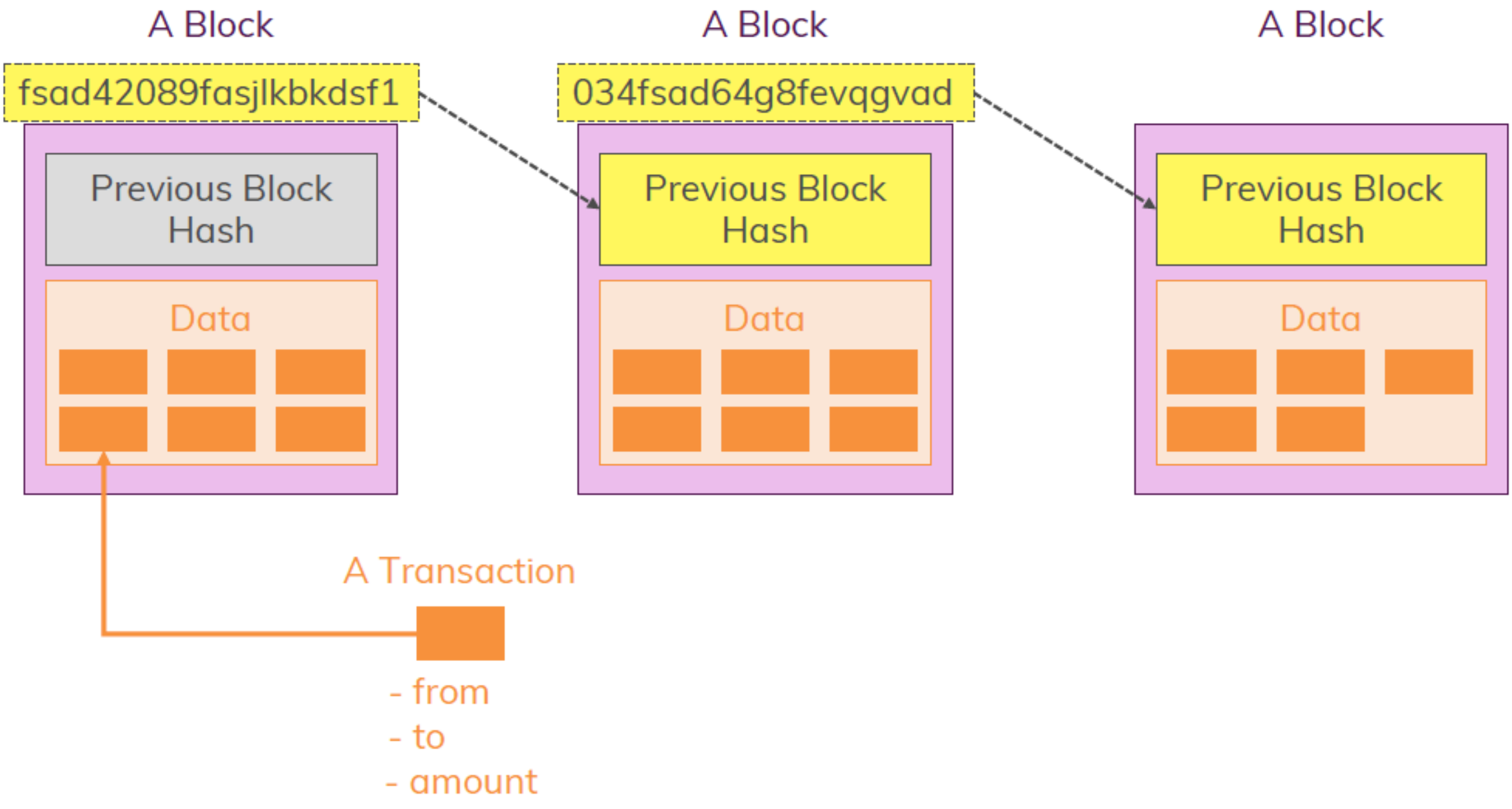
Bitcoin solves the double-spending problem in digital cash with a decentralized network, i.e. the Blockchain.

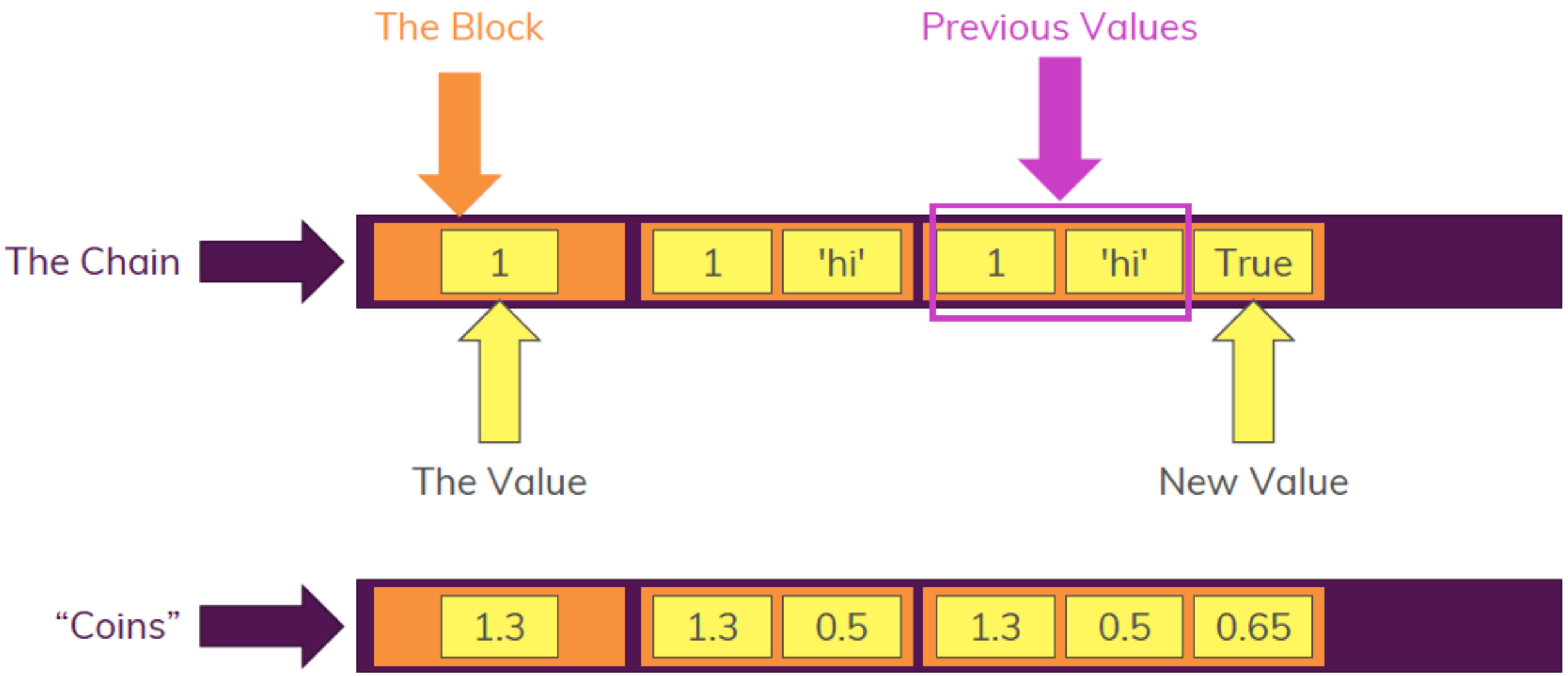
Solución 1: confiar en una entidad centralizada (como un banco)

Solución 2: implementar una solución descentralizada (como la blockchain)

*En 2008, Satoshi Nakamoto, en un texto llamado "*Bitcoin: A Peer-to-Peer Electronic Cash System*", propone una solución descentralizada con las siguientes características:*

- *es distribuida: el libro de cuentas (el libro mayor) está replicado en varias computadoras, en lugar de en un servidor central. Cualquiera puede tener una copia completa de la* blockchain.
- *es criptográfica: la criptografía es empleada para asegurar que el remitente posee los fondos que intenta enviar y para decidir cómo se agregan las transacciones a la* blockchain.
- *es inmutable: solo se pueden agregar transacciones a la* blockchain*, no se pueden modificar o eliminar transacciones.*
- *usa prueba de trabajo (Proof of Work, PoW): un tipo especial de participantes en la red, llamados* ***mineros****, compiten para encontrar la solución de un problema criptográfico que les permitirá agregar un bloque de transacciones a la* blockchain*. Esto le permite al sistema ser seguro.*


A Block
fsad42089fasjlkbkdsf1
Previous Block Hash
Data
A Block
034fsad64g8fevqgvad
Previous Block Hash
Data
A Block
Previous Block Hash
Data
A Transaction
- from
- to
- amount


The Block
Previous Values
The Chain
1
1
'hi'
1
'hi'
True
The Value
New Value
"Coins"
1.3
1.3
0.5
1.3
0.5
0.65

BLOCKCHAIN DE JUGUETE

Implementemos en Python una blockchain que solo satisfaga ser distribuida e inmutable. Más adelante que tengamos más tecnología implementaremos una blockchain criptográfica y con PoW.

Creamos un archivo blockchain.py

blockchain.py

```
blockchain = []

def add_value():
    blockchain.append(5.3)
    print(blockchain)

add_value()
add_value()
add_value()
```

Vemos que sólo se agrega el valor 5.3 varias veces a la lista.

Probemos con

blockchain.py

```
blockchain = [1]

def add_value():
    blockchain.append([blockchain[0], 5.3])
    print(blockchain)

add_value()
add_value()
add_value()
```

```
1. [1]
2. [1] <— [1,5.3]
3. [1,[1,5.3]] <— [1,5.3]
4. [1,[1,5.3],[1,5.3]] <— [1,5.3]
5. [1,[1,5.3],[1,5.3],[1,5.3]]
```

Ahora con

blockchain.py

```
blockchain = [1]

def add_value():
    blockchain.append([blockchain[-1], 5.3])
    print(blockchain)

add_value()
add_value()
add_value()
```

```
1. [1]
2. [1] <— [1,5.3]
3. [1,[1,5.3]] <— [[1,5.3],5.3]
4. [1,[1,5.3],[[1,5.3],5.3]] <— [[[1,5.3],5.3],5.3]
5. [1,[1,5.3],[[1,5.3],5.3],[[[1,5.3],5.3],5.3]]
```

Modificamos para que todas las entradas sean listas:

blockchain.py

```
blockchain = [[1]]

def add_value():
    blockchain.append([blockchain[-1], 5.3])
    print(blockchain)

add_value()
add_value()
add_value()
```

```
1. [[1]]
2. [[1]] <— [[1],5.3]
3. [[1], [[1],5.3]] <— [[[1],5.3],5.3]
4. [[1], [[1],5.3], [[[1],5.3],5.3]] <— [[[1],5.3], [[[1],5.3],5.3],5.3]
5. [[1], [[1],5.3], [[[1],5.3],5.3], [[[1],5.3], [[[1],5.3],5.3],5.3]]
```

FUNCIONES CON ARGUMENTOS

La función que definimos sólo es capaz de agregar una cantidad fija de bitcoins cada vez. Sería mejor que fuera capaz de agregar una cantidad que nosotros elijamos. Veamos un ejemplo de cómo se hace esto.

```
def saludar(nombre):
    print('hola ' + nombre)

saludar('pedro')
saludar(‘alberto')
```

blockchain.py

```
blockchain = [[1]]

def add_value(transaction_amount):
    blockchain.append([blockchain[-1], transaction_amount])
    print(blockchain)

add_value(2)
add_value(9)
add_value(10.89)
```

FUNCIONES QUE DEVUELVEN VALORES

Además de ejecutar código, las funciones pueden devolver valores como resultado de su ejecución. Veamos un ejemplo:

```
def suma(a, b):
    return a + b

print(suma(1,2))
```

Las funciones pueden tener argumentos predeterminados:

```
def suma(a, b=0):
    return a + b

print(suma(1,2))
print(sum(1))
```

blockchain.py

```
blockchain = [[1]]

def get_last_blockchain_value():
    return blockchain[-1]

def add_value(transaction_amount):
    blockchain.append([get_last_blockchain_value(), transaction_amount])
    print(blockchain)

add_value(2)
add_value(9)
add_value(10.89)
```

blockchain.py

```
blockchain = []

def get_last_blockchain_value():
    return blockchain[-1]

def add_value(transaction_amount, last_transaction=[1]):
    blockchain.append([last_transaction, transaction_amount])

add_value(2)
add_value(last_transaction=get_last_blockchain_value(),
          transaction_amount=9)
add_value(10.89, get_last_blockchain_value())

print(blockchain)
```

blockchain.py

```
blockchain = []

def get_last_blockchain_value():
    return blockchain[-1]

def add_value(transaction_amount, last_transaction=[1]):
    blockchain.append([last_transaction, transaction_amount])

tx_amount = float(input('El monto de su transacción, por favor'))
add_value(tx_amount)

tx_amount = float(input('El monto de su transacción, por favor'))
add_value(last_transaction=get_last_blockchain_value(),
          transaction_amount=tx_amount)

tx_amount = float(input('El monto de su transacción, por favor'))
add_value(tx_amount, get_last_blockchain_value())

print(blockchain)
```

blockchain.py

```
blockchain = []

def get_last_blockchain_value():
    return blockchain[-1]

def add_value(transaction_amount, last_transaction=[1]):
    blockchain.append([last_transaction, transaction_amount])

def get_user_input():
    return float(input('El monto de su transacción, por favor'))

tx_amount = get_user_input()
add_value(tx_amount)

tx_amount = get_user_input()
add_value(last_transaction=get_last_blockchain_value(),
          transaction_amount=tx_amount)

tx_amount = get_user_input()
add_value(tx_amount, get_last_blockchain_value())

print(blockchain)
```

ALCANCE DE VARIABLES

Alcance global

```
nombre = 'Pedro'
def saludar():
    print('hola ' + nombre)

saludar()
```

Alcance local

```
nombre = 'Pedro'
def saludar():
    texto = 'buenas tardes'
    print('hola ' + nombre + ', ' + texto)

saludar()
print(nombre)
print(texto)
```

COMENTARIOS

blockchain.py

```
# initialize blockchain list
blockchain = []

def get_last_blockchain_value():
    """ Returns last value from blockchain"""
    return blockchain[-1]

def add_value(transaction_amount, last_transaction=[1]):
    blockchain.append([last_transaction, transaction_amount])

def get_user_input():
    return float(input('El monto de su transacción, por favor'))

tx_amount = get_user_input()
add_value(tx_amount)

tx_amount = get_user_input()
add_value(last_transaction=get_last_blockchain_value(),
          transaction_amount=tx_amount)

tx_amount = get_user_input()
add_value(tx_amount, get_last_blockchain_value())

print(blockchain)
```

BUCLES Y CONDICIONALES

`for`

```
for element in list:
    print(element)
```

Un bucle `for` permite iterar por los elementos de un iterable (p. ej. una lista)

`while`

```
while True:
    print('Infinity')
```

Un bucle `while` permite repetir una ejecución mientras la condición sea verdadera.

`break` nos permite salir del bucle antes de que este finalice.
`continue` nos permite saltar una iteración.

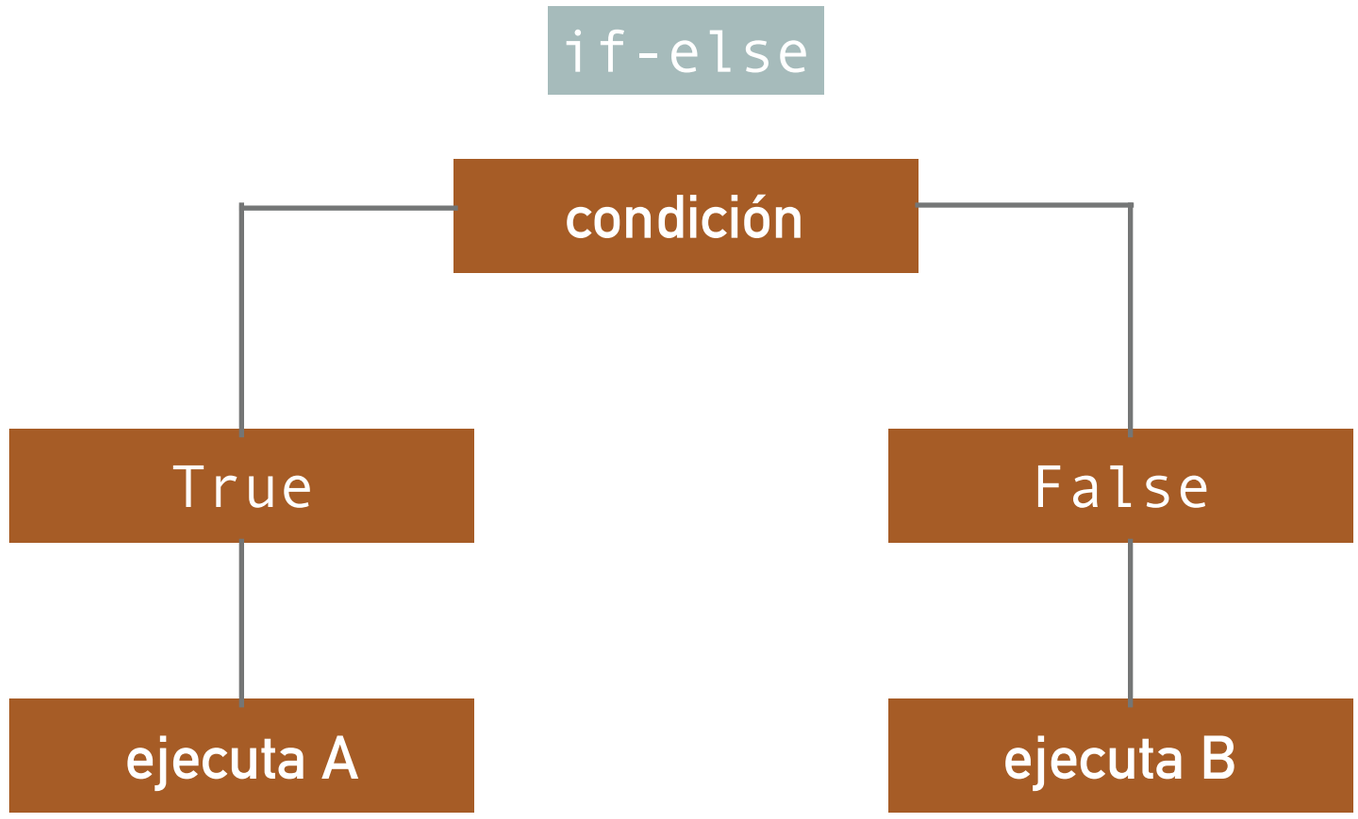
blockchain.py

```
# initialize blockchain list
blockchain = []

def get_last_blockchain_value():
    """ Returns last value from blockchain"""
    return blockchain[-1]

def add_value(transaction_amount, last_transaction=[1]):
    blockchain.append([last_transaction, transaction_amount])

def get_user_input():
    return float(input('El monto de su transacción, por favor'))

tx_amount = get_user_input()
add_value(tx_amount)

tx_amount = get_user_input()
add_value(last_transaction=get_last_blockchain_value(),
          transaction_amount=tx_amount)

tx_amount = get_user_input()
add_value(tx_amount, get_last_blockchain_value())

for block in blockchain:
    print(block)
```

blockchain.py

```
# initialize blockchain list
blockchain = []

def get_last_blockchain_value():
    """ Returns last value from blockchain"""
    return blockchain[-1]

def add_value(transaction_amount, last_transaction=[1]):
    blockchain.append([last_transaction, transaction_amount])

def get_user_input():
    return float(input('El monto de su transacción, por favor'))

tx_amount = get_user_input()
add_value(tx_amount)

while True:
    tx_amount = get_user_input()
    add_value(tx_amount, get_last_blockchain_value())

    for block in blockchain:
        print(block)
```

CONDICIONALES




```
if condition:
    do_a()
```

```
if condition:
    do_a()
else:
    do_b()
```

```
if condition:
    do_a()
elif other_condition:
    do_b()
else:
    do_c()
```

blockchain.py

```
# initialize blockchain list
blockchain = []

def get_last_blockchain_value():
    """ Returns last value from blockchain"""
    return blockchain[-1]

def add_value(transaction_amount, last_transaction=[1]):
    blockchain.append([last_transaction, transaction_amount])

def get_transaction_value():
    return float(input('El monto de su transacción, por favor'))

def get_user_choice():
    return input('Su elección')

def print_blockchain_elements():
    for block in blockchain:
            print(block)

tx_amount = get_user_input()
add_value(tx_amount)

while True:
    print('Elija por favor:’)
    print('1: Agrege un nuevo valor de transacción')
    print('2: Muestre los bloques de la cadena’)
    user_choice = get_user_choice()
    if user_choice == ‘1':
        tx_amount = get_transaction_value()
        add_value(tx_amount, get_last_blockchain_value())
    else:
        print_blockchain_elements()
```

blockchain.py

```
. . .

while True:
    print('Elija por favor:’)
    print('1: Agrege un nuevo valor de transacción')
    print('2: Muestre los bloques de la cadena’)
    user_choice = get_user_choice()
    if user_choice == '1':
        tx_amount = get_transaction_value()
        add_value(tx_amount, get_last_blockchain_value())
    elif user_choice == '2':
        print_blockchain_elements()
    else:
        print('Opción inválida, elija una opción de la lista')

. . .
```

blockchain.py

```
. . .

while True:
    print('Elija por favor:’)
    print('1: Agregue un nuevo valor de transacción')
    print('2: Muestre los bloques de la cadena’)
    print('s: Salir')
    user_choice = get_user_choice()
    if user_choice == '1':
        tx_amount = get_transaction_value()
        add_value(tx_amount, get_last_blockchain_value())
    elif user_choice == '2':
        print_blockchain_elements()
    elif user_choice == 's':
        break
    else:
        print('Opción inválida, elija una opción de la lista')

. . .
```

MEJORANDO EL CÓDIGO CON BUCLES Y CONDICIONALES

blockchain.py

```
. . .

def get_last_blockchain_value():
    """ Returns last value from blockchain"""
    if len(blockchain) < 1:
        return None
    return blockchain[-1]


. . .
```

blockchain.py

```
. . .

def add_transaction(transaction_amount, last_transaction=[1]):
    if last_transaction == None:
        last_transaction = [1]
    blockchain.append([last_transaction, transaction_amount])


. . .
```

Podemos eliminar el código en que pedimos, por primera vez, un valor de transacción.

blockchain.py

```
. . .
def print_blockchain_elements():
    for block in blockchain:
            print(block)

while True:
    print('Elija por favor:’)
    print('1: Agrege un nuevo valor de transacción')
    print('2: Muestre los bloques de la cadena’)
    user_choice = get_user_choice()
    if user_choice == ‘1':
        tx_amount = get_transaction_value()
        add_value(tx_amount, get_last_blockchain_value())
    else:
        print_blockchain_elements()
. . .
```

BOOLEANOS

Verdadero o Falso

True

False

Es usado en pruebas condicionales (if)

Es el resultado de operaciones lógicas

==

!=

>=

is

>

<

<=

in

```
1 == 1 # True
1 != 1 # False
```

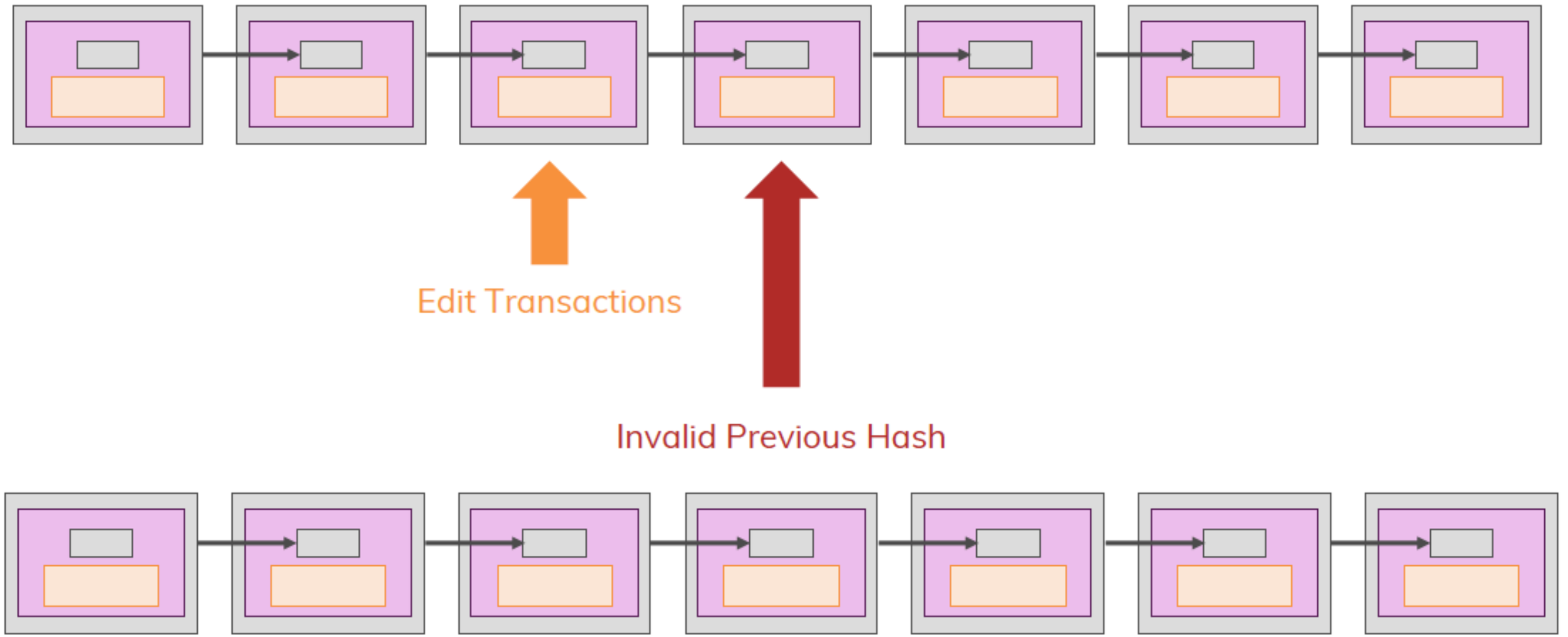
```
edad = 34
edad != 35 # True
edad == 34 # True
```

```
data = [1,2,3]
1 in data # True
4 in data # False
4 not in data # True
```

```
edad = 34
(edad > 30) and (edad < 40) # True
```

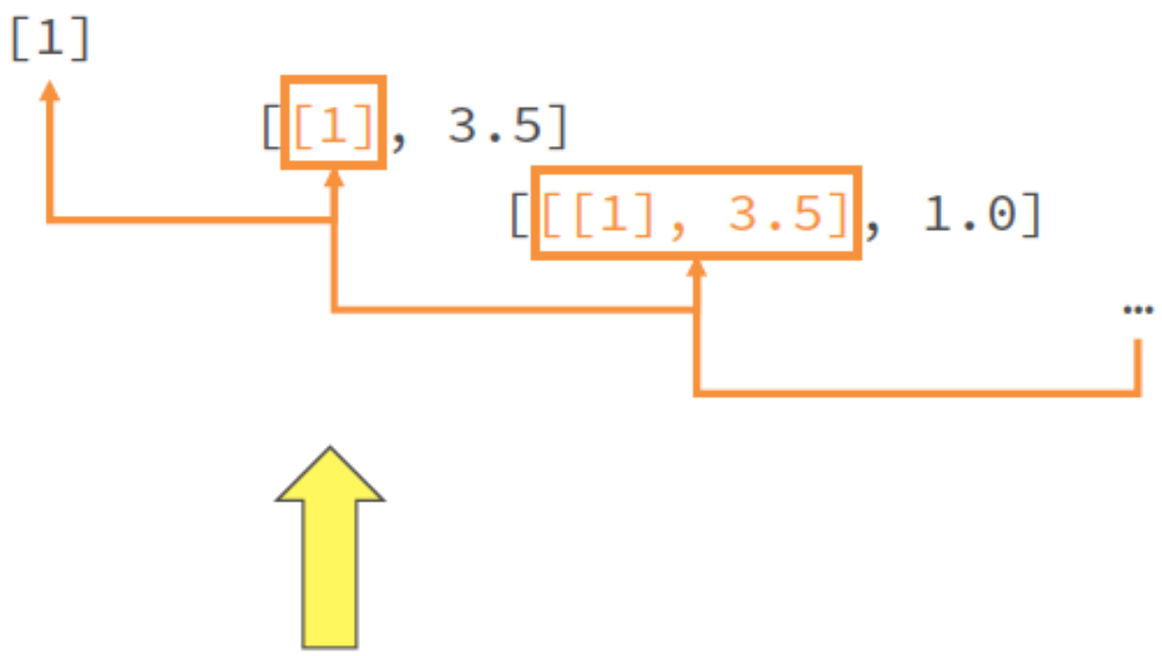
blockchain.py

```
. . .

while True:
    print('Elija por favor:’)
    print('1: Agrege un nuevo valor de transacción')
    print('2: Muestre los bloques de la cadena’)
    print('h: Manipular la blockchain')
    print('s: Salir')
    user_choice = get_user_choice()
    if user_choice == '1':
        tx_amount = get_transaction_value()
        add_value(tx_amount, get_last_blockchain_value())
    elif user_choice == '2':
        print_blockchain_elements()
    elif user_choice == 'h':
        if len(blockchain) >= 1:
            blockchain[0] = [2]
    elif user_choice == 's':
        break
    else:
        print('Opción inválida, elija una opción de la lista')

. . .
```



Edit Transactions

Invalid Previous Hash



blockchain.py

```
. . .
def verify_chain():
    block_index = 0
    is_valid = True
    for block in blockchain:
        if block_index == 0:
            block_index = block_index + 1
            continue
        if block[0] == blockchain[block_index - 1]:
            is_valid = True
        else:
            is_valid = False
            break
        block_index = block_index + 1
    return is_valid
. . .
```

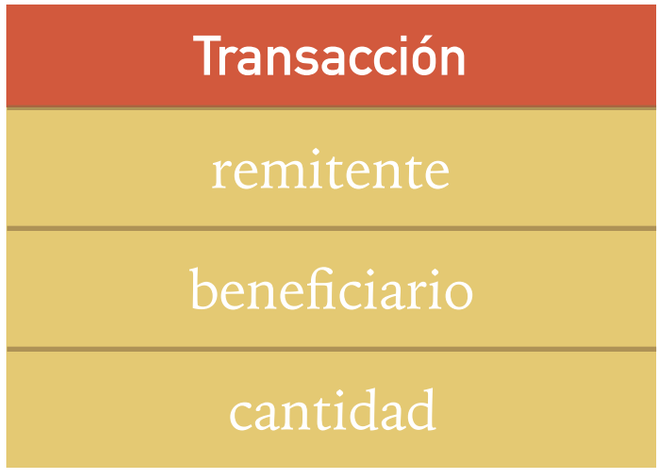
blockchain.py

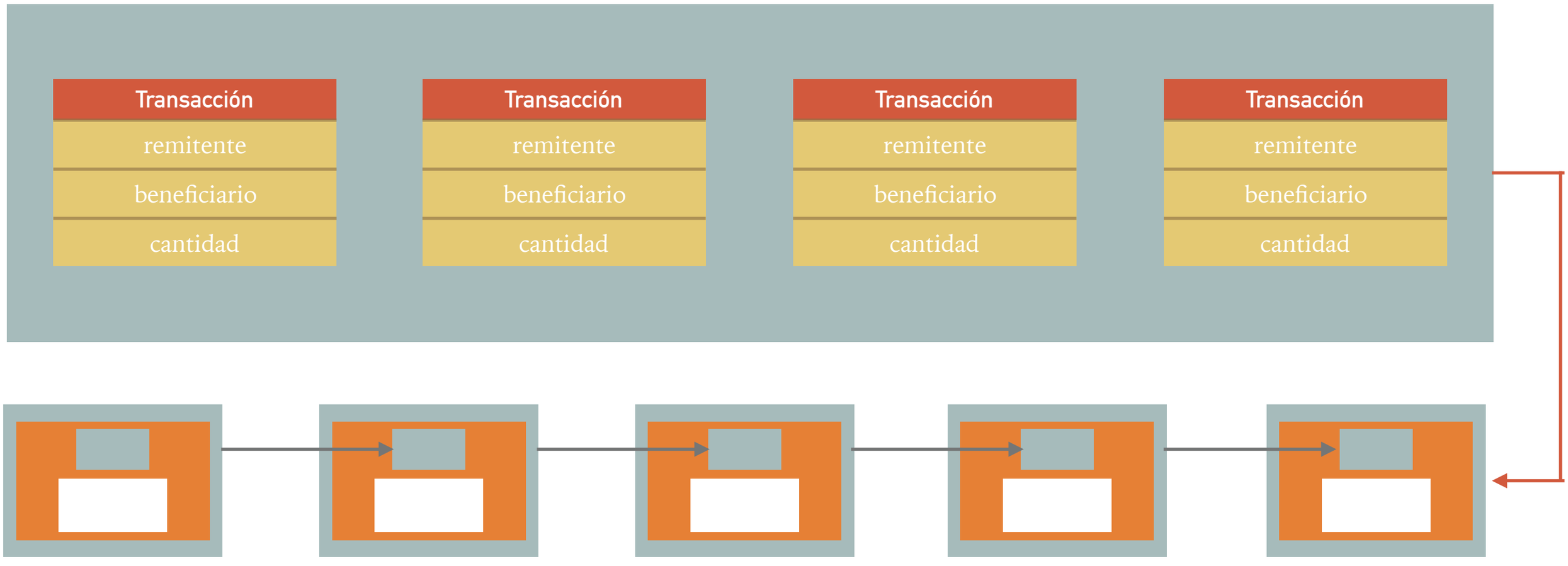
```
. . .

while True:
    print('Elija por favor:’)
    print('1: Agregue un nuevo valor de transacción')
    print('2: Muestre los bloques de la cadena’)
    print('h: Manipular la blockchain')
    print('s: Salir')
    user_choice = get_user_choice()
    if user_choice == '1':
        tx_amount = get_transaction_value()
        add_value(tx_amount, get_last_blockchain_value())
    elif user_choice == '2':
        print_blockchain_elements()
    elif user_choice == 'h':
        if len(blockchain) >= 1:
            blockchain[0] = [2]
    elif user_choice == 's':
        break
    else:
        print('Opción inválida, elija una opción de la lista’)
    if not verify_chain():
        print('Blockchain inválida')
        break

. . .
```

blockchain.py

```
. . .

waiting_for_input = True

while waiting_for_input:
    print('Elija por favor:’)
    print('1: Agrege un nuevo valor de transacción')
    print('2: Muestre los bloques de la cadena’)
    print('h: Manipular la blockchain')
    print('s: Salir')
    user_choice = get_user_choice()
    if user_choice == '1':
        tx_amount = get_transaction_value()
        add_value(tx_amount, get_last_blockchain_value())
    elif user_choice == '2':
        print_blockchain_elements()
    elif user_choice == 'h':
        if len(blockchain) >= 1:
            blockchain[0] = [2]
    elif user_choice == 's':
        waiting_for_input = False
    else:
        print('Opción inválida, elija una opción de la lista')
    if not verify_chain():
        print('Blockchain inválida')
        break
else:
    print('El usuario ha finalizado las transacciones')

. . .
```

ESTRUCTURAS DE DATOS MÁS COMPLEJAS





Bloque
Hash
Índice
Transacciones

Los participantes solo pueden aparecer una vez.

Remitentes y beneficiarios
Pedro
Ximena
Luis
Ana

Necesitamos las siguientes estructuras de datos:

a. Transacción*: parejas* key-value*, no importa el orden;*

b. Bloque*: lista mutable (editable) de valores, no importa el orden;*

c. Blockchain*: lista mutable (editable) de valores, el orden importa;*

d. Participantes: lista mutable (editable) de valores únicos, no importa el orden.

ITERABLES

Lista `list`	`['pedro', 'ximena']`	Mutable, lista ordenada, duplicados son permitidos, principalmente un sólo tipo de datos
Conjunto `set`	`{'pedro', 'ximena'}`	Mutable, lista no ordenada, sin duplicados, principalmente un sólo tipo de datos
Tuplos `tuple`	`('pedro', 'ximena')`	Inmutable, lista ordenada, duplicados son permitidos, principalmente varios tipos de datos
Diccionario `dict`	`{'nombre': 'pedro', 'n':2}`	Mutable, mapeo no ordenado, sin keys duplicadas, varios tipos de datos

ESTRUCTURAS DE DATOS MÁS COMPLEJAS

diccionario

Transacción

remitente

beneficiario

cantidad

lista

Transacción

remitente

beneficiario

cantidad

Transacción

remitente

beneficiario

cantidad

Transacción

remitente

beneficiario

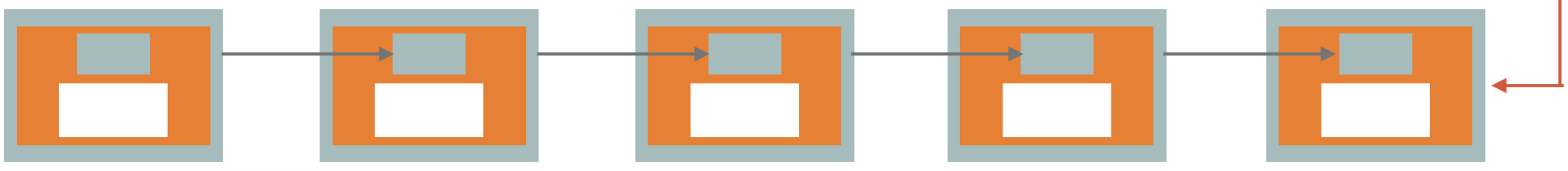
cantidad

Transacción

remitente

beneficiario

cantidad



lista

diccionario

Bloque
Hash
Índice
Transacciones

conjunto

Remitentes y beneficiarios
Pedro
Ximena
Luis
Ana

Una transacción típicamente contiene:
- quien envía las monedas
- quien recibe las monedas
- cuantas monedas se envían

Transacción
remitente
beneficiario
cantidad

Transacción
remitente
beneficiario
cantidad

Transacción
remitente
beneficiario
cantidad

Transacción
remitente
beneficiario
cantidad

Las monedas que pertenecen a un bloque no se ha integrado a la blockchain son transacciones abiertas. Para integrar las transacciones abiertas a la blockchain, es necesario “procesarlas”.

blockchain.py

```
. . .

open_transactions = []

def mine_block():
    pass

def add_transaction(sender, recipient, amount=1.0):
    transaction = {
        'sender': sender,
        'recipient': recipient,
        'amount': amount
    }
    open_transactions.append(transaction)
. . .
```

blockchain.py

```
. . .

owner = 'Pedro'

def get_transaction_value():
    tx_recipient = input('Escriba el beneficiario de la transacción: ‘)
    tx_amount = float(input('El monto de la transacción, por favor: '))
    return (tx_recipient, tx_amount)


. . .
```

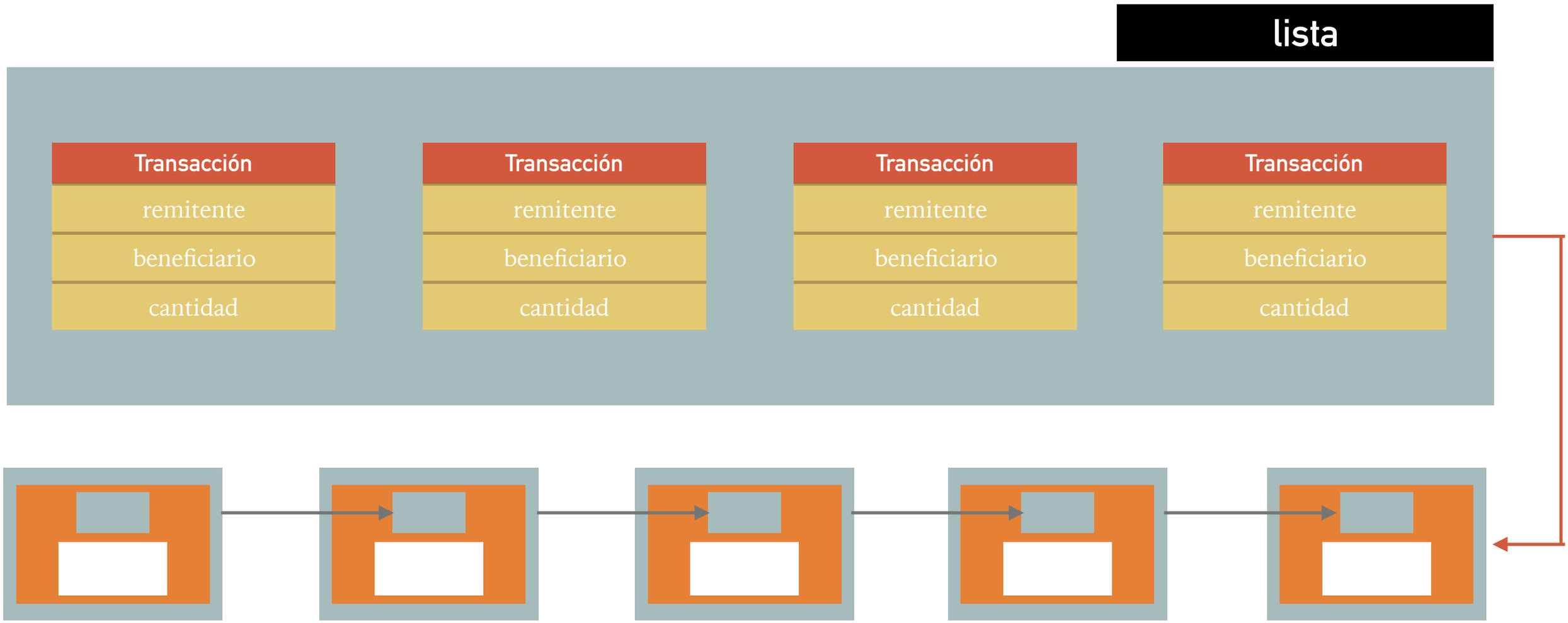
blockchain.py

```
. . .

def add_transaction(recipient, sender=owner, amount=1.0):
    transaction = {
        'sender': sender,
        'recipient': recipient,
        'amount': amount
    }
    open_transactions.append(transaction)


. . .
```

blockchain.py

```
. . .

waiting_for_input = True

while waiting_for_input:
    print('Elija por favor:’)
    print('1: Agrege un nuevo valor de transacción')
    print('2: Muestre los bloques de la cadena’)
    print('h: Manipular la blockchain')
    print('s: Salir')
    user_choice = get_user_choice()
    if user_choice == '1':
        tx_data = get_transaction_value()
        recipient, amount = tx_data
        add_transaction(recipient, amount=amount)
        print(open_transactions)
    elif user_choice == '2':
        print_blockchain_elements()
    elif user_choice == 'h':
        if len(blockchain) >= 1:
            blockchain[0] = [2]
    elif user_choice == 's':
        waiting_for_input = False
    else:
        print('Opción inválida, elija una opción de la lista')
    if not verify_chain():
        print('Blockchain inválida')
        break
else:
    print('El usuario ha finalizado las transacciones')
```

MINANDO BLOQUES



lista

blockchain.py

```
. . .

genesis_block = {
    'previous_hash': '',
    'index': 0,
    'transactions': []
}
blockchain.append(genesis_block)

def mine_block():
    last_block = blockchain[-1]
    block = {
        'previous_hash': ‘XYZ',
        'index': len(blockchain),
        'transactions': open_transactions
    }
    blockchain.append(block)

. . .
```

blockchain.py

```
. . .

def mine_block():
    last_block = blockchain[-1]
    hashed_block = ''
    for key in last_block:
        value = last_block[key]
        hashed_block = hashed_block + str(value)

    print(hashed_block)

    block = {
        'previous_hash': 'XYZ',
        'index': len(blockchain),
        'transactions': open_transactions
    }
    blockchain.append(block)

. . .
```

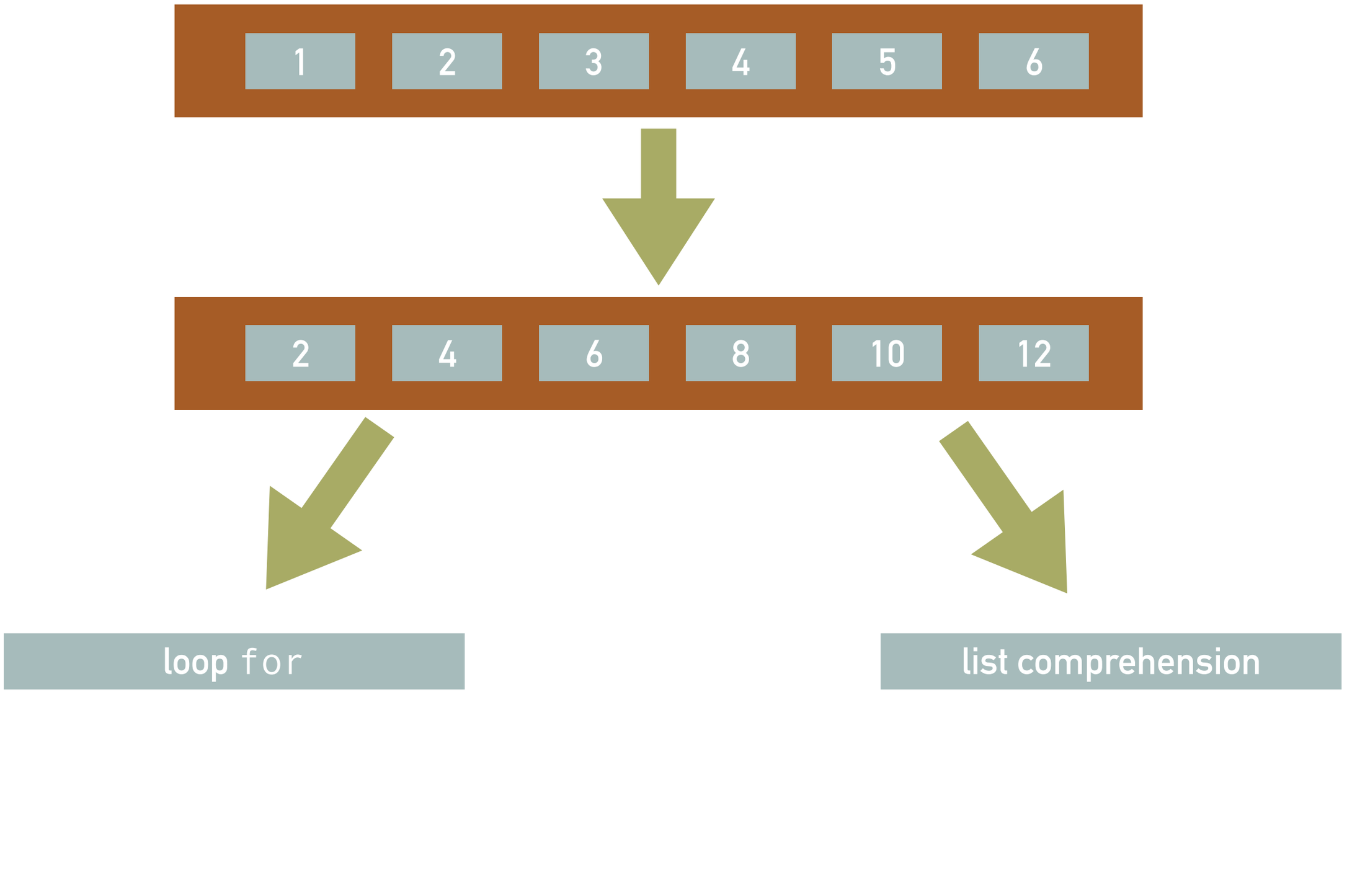
blockchain.py

```
. . .

while waiting_for_input:
    print('Elija por favor:’)
    print('1: Agrege un nuevo valor de transacción’)
    print('2: Minar bloque')
    print('3: Muestre los bloques de la cadena’)
    print('h: Manipular la blockchain')
    print('s: Salir')
    user_choice = get_user_choice()
    if user_choice == '1':
        tx_data = get_transaction_value()
        recipient, amount = tx_data
        add_transaction(recipient, amount=amount)
        print(open_transactions)
    elif user_choice == 2:
        mine_block()
    elif user_choice == '3':
        print_blockchain_elements()
    elif user_choice == 'h':
        if len(blockchain) >= 1:
            blockchain[0] = [2]
    elif user_choice == 's':
        waiting_for_input = False
    else:
        print('Opción inválida, elija una opción de la lista')
    #if not verify_chain():
    #    print('Blockchain inválida')
    #    break
else:
    print('El usuario ha finalizado las transacciones')
```

blockchain.py

```
. . .

def mine_block():
    last_block = blockchain[-1]
    hashed_block = ''
    for key in last_block:
        value = last_block[key]
        hashed_block = hashed_block + str(value)

    print(hashed_block)

    block = {
        'previous_hash': hashed_block,
        'index': len(blockchain),
        'transactions': open_transactions
    }
    blockchain.append(block)

. . .
```

LIST COMPREHENSIONS



```
>>> simple_list = [1,2,3,4]
>>> double_list = []
>>> for element in simple_list:
...     double_list.append(element * 2)
...
>>> double_list
[2, 4, 6, 8]
>>> double_list = []
>>> double_list = [el * 2 for el in simple_list]
>>> double_list
[2, 4, 6, 8]
```

blockchain.py

```
. . .

def mine_block():
    last_block = blockchain[-1]
    hashed_block = str([last_block[key] for key in last_block])

    print(hashed_block)

    block = {
        'previous_hash': hashed_block,
        'index': len(blockchain),
        'transactions': open_transactions
    }
    blockchain.append(block)

. . .
```

blockchain.py

```
. . .

def mine_block():
    last_block = blockchain[-1]
    hashed_block = '-'.join([str(last_block[key]) for key in last_block])

    print(hashed_block)

    block = {
        'previous_hash': hashed_block,
        'index': len(blockchain),
        'transactions': open_transactions
    }
    blockchain.append(block)

. . .
```

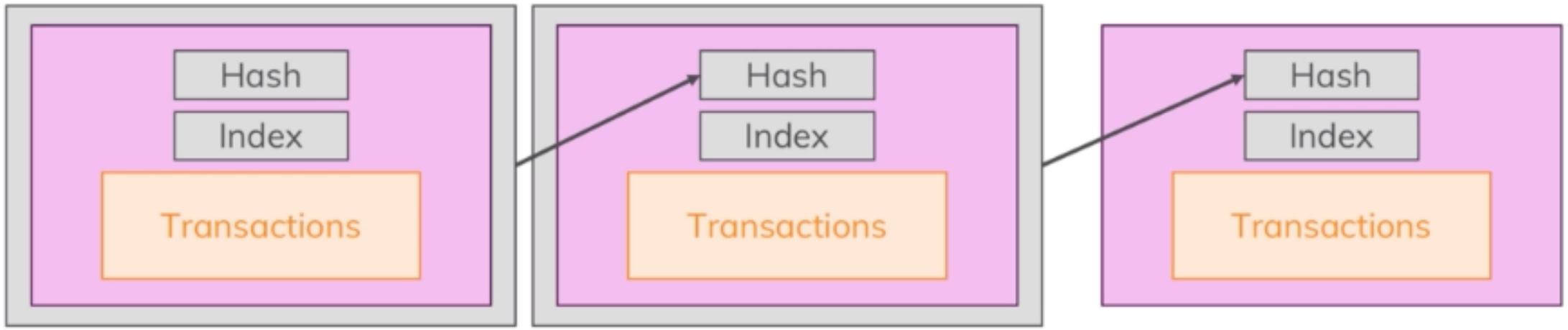
DICT COMPREHENSIONS

```
>>> stats = [('age', 29), ('weight',178)]
>>> dict_stats = {key: value for key, value in stats}
>>> dict_stats
{'age': 29, 'weight': 178}
```

LIST COMPREHENSIONS CON IF

```
>>> simple_list = [1,2,3,4]
>>> dup_list = [el * 2 for el in simple_list if el % 2 == 0]
>>> dup_list
[4, 8]
>>> calc_items = [1, 2]
>>> dup_list = [el * 2 for el in simple_list if el in calc_items]
>>> dup_list
[2, 4]
```

VALIDACIÓN DE BLOQUES



blockchain.py

```
. . .

def hash_block(block):
    return '-'.join([str(block[key]) for key in block])

def mine_block():
    last_block = blockchain[-1]
    hashed_block = hash_block(last_block)

    print(hashed_block)

    block = {
        'previous_hash': hashed_block,
        'index': len(blockchain),
        'transactions': open_transactions
    }
    blockchain.append(block)

. . .
```

blockchain.py

```
. . .

def verify_chain():
    for index, block in enumerate(blockchain):
        if index == 0:
            continue
        if block['previous_hash'] != hash_block(blockchain[index - 1])
            return False
    return True

. . .
```

blockchain.py

```
. . .

while waiting_for_input:
    print('Elija por favor:’)
    print('1: Agrege un nuevo valor de transacción’)
    print('2: Minar bloque')
    print('3: Muestre los bloques de la cadena’)
    print('h: Manipular la blockchain')
    print('s: Salir')
    user_choice = get_user_choice()
    if user_choice == '1':
        tx_data = get_transaction_value()
        recipient, amount = tx_data
        add_transaction(recipient, amount=amount)
        print(open_transactions)
    elif user_choice == 2:
        mine_block()
    elif user_choice == '3':
        print_blockchain_elements()
    elif user_choice == 'h':
        if len(blockchain) >= 1:
            blockchain[0] = {
                'previous_hash': '',
                'index': 0,
                'transactions': [{'sender': 'Ximena', 'recipient': 'Pedro', 'amount': 10}]
            }
    elif user_choice == 's':
        waiting_for_input = False
    else:
        print('Opción inválida, elija una opción de la lista')
    if not verify_chain():
        print('Blockchain inválida')
        break
else:
    print('El usuario ha finalizado las transacciones')
```