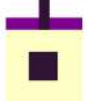
Curso de

Estructuras de Datos Lineales con Python

Héctor Vega @TerranigmArk

Un poco sobre mí

- 👨‍💼 Lic. en Administración.
- 🤓 Apasionado a la tecnología.
- 💻 En compañías tech desde 2015.
- 👨‍🏫 Course Director en Platzi.
- 🐍 Pythonista de corazón.



Ahora sobre ti

- 🐍 Pythonista de corazón.
- 🧠 Piensas de forma lógica.
- 🤔 Conoces lo esencial de Python.
- 🐧 La terminal es tu amiga.
- 💪 Deseas crear algo genial.

Lo que debes saber

Pensamiento Lógico y Algoritmos

Terminal y Línea de Comandos

Lo que debes saber

Python Básico

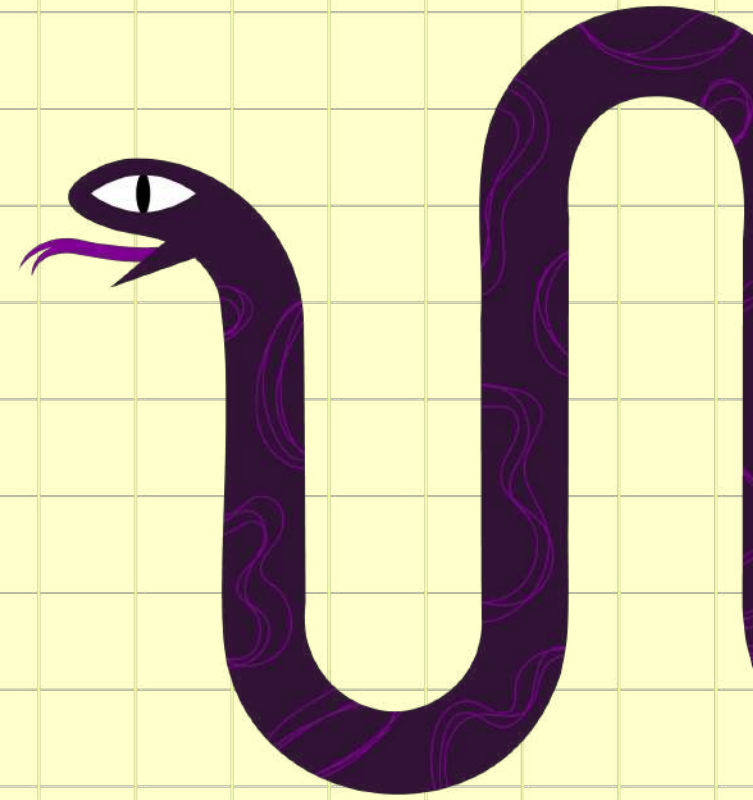
Python Intermedio

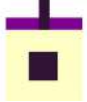
Lo que debes saber

Pensamiento Computacional

Programación Orientada a Objetos

Objetivos

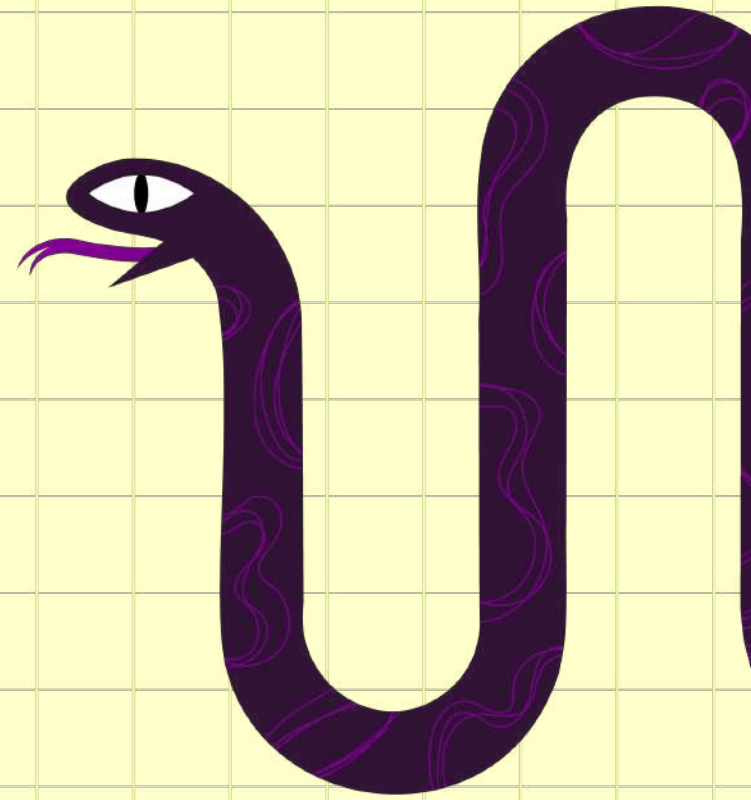




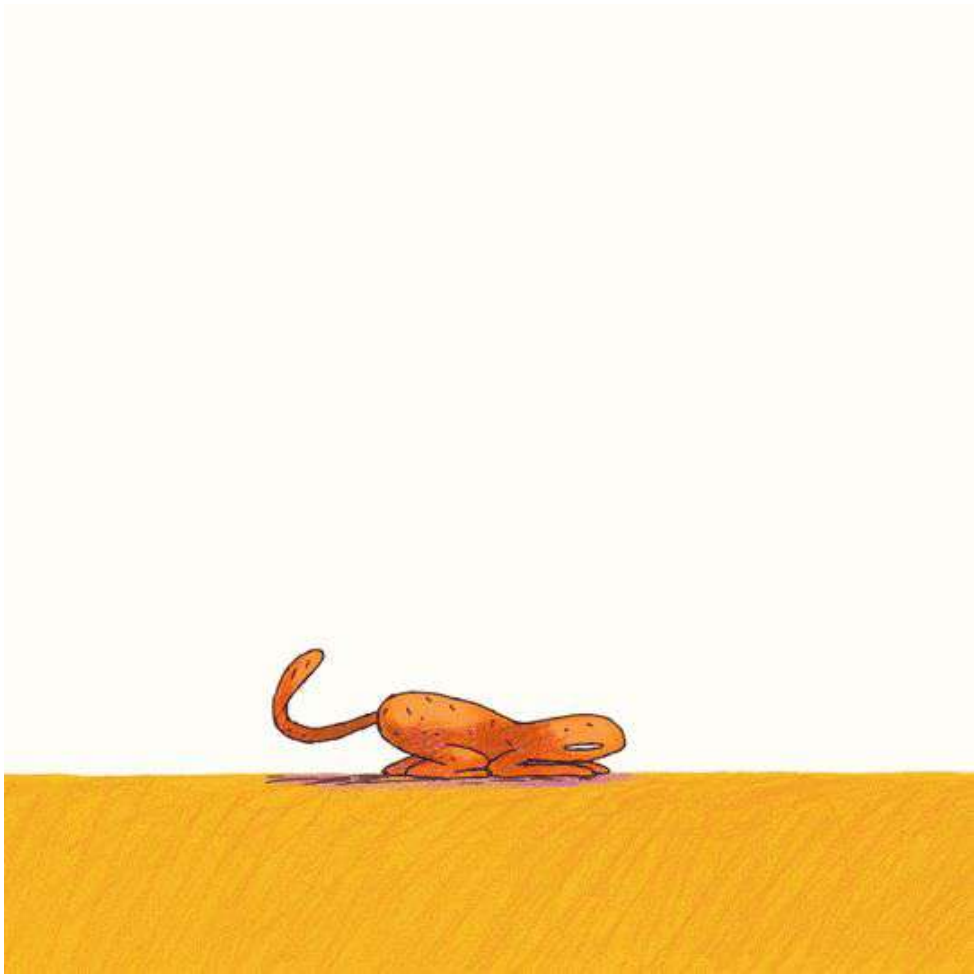
De aprendizaje

- Entender el concepto e importancia de estructuras de datos.
- Comprender el comportamiento, uso e implementación de estructuras de datos lineales con Python.
- Poner en práctica lo aprendido.

¿Por qué Python?



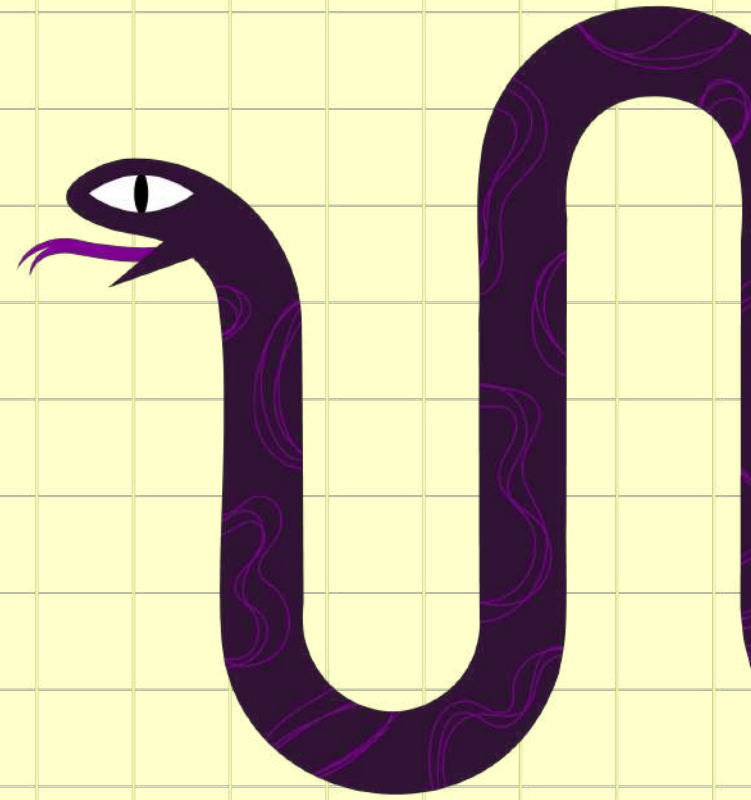
Evolución



Lenguaje genial

- Sintaxis clara y simple
- Semántica segura
- Escalable
- Interactivo
- Propósito general
- Gratis y popular

Agenda

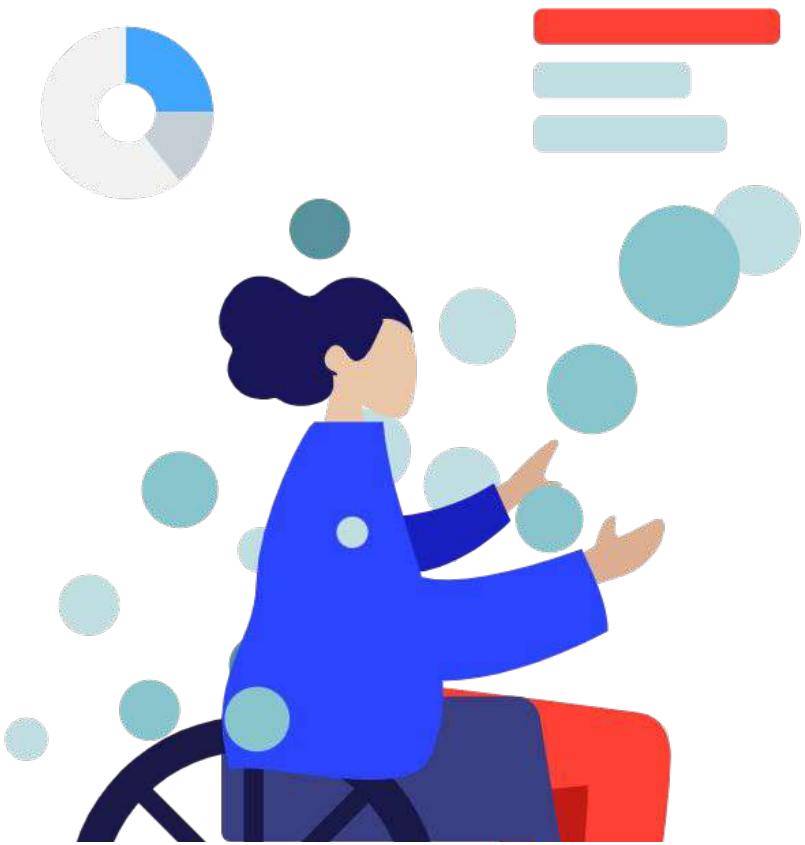


Introducción

- Introducción
- Elementos en Python
- Colecciones
- Tipos y operaciones

Estructuras de datos lineales

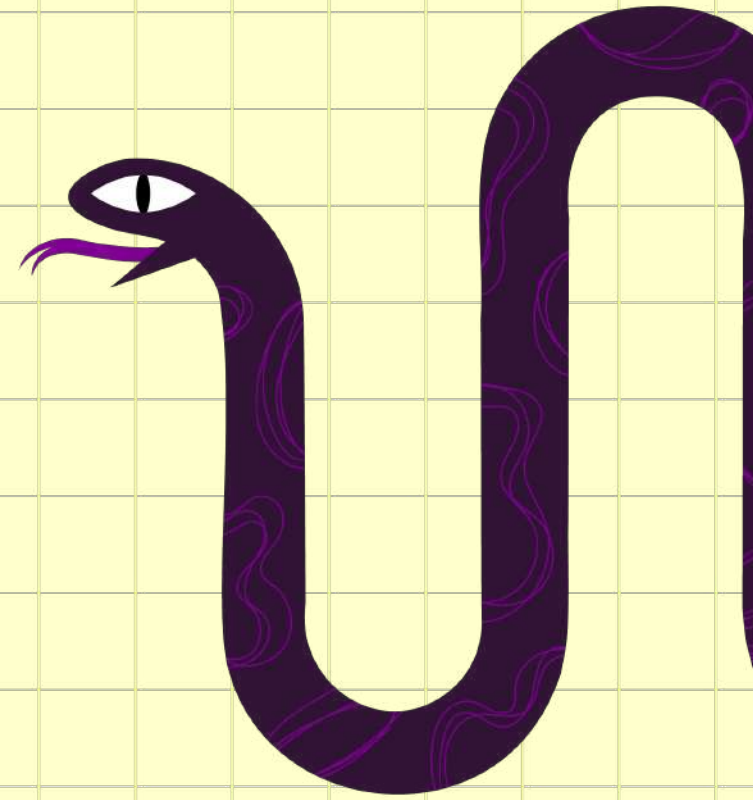
- Arrays
- Nodes
- Linked lists
- Stacks
- Queues



Elementos de la programación en Python

¡Ya sabes programar en Python!

¿Qué sigue?



```
"""
Guess a number game with the user.
"""

import random

def main():
    """Input the upper and lower limit of a range of numbers
    and lets the user guess the random number until correct.
    """
    lower_limit = int(input("Enter the smaller number of the range: "))
    upper_limit = int(input("Enter the larger number of the range: "))
    secret_number = random.randint(lower_limit, upper_limit)
    tries = 0

    while True:
        tries += 1
        user_number = int(input("Enter a number to try to guess: "))

        if user_number < secret_number:
            print("Try with a higher number.")
        elif user_number > secret_number:
            print("Try with a lower number.")
        else:
            print(f"You won! It took you {tries} tries")
            break

if __name__ == "__main__":
    main()
```

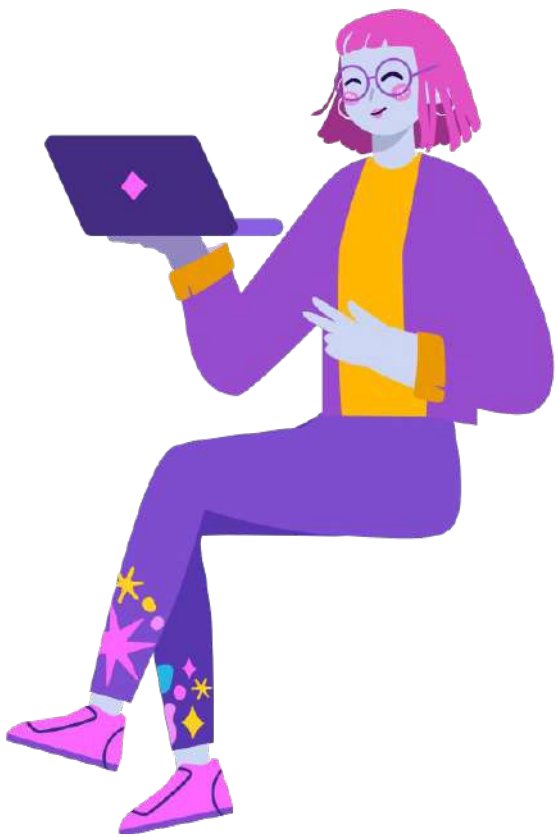
A tomar en cuenta

Elementos léxicos

- if, while, def, etc.

Convenciones

- variable
- CONSTANTE
- nombre_funcion
- nombreClase

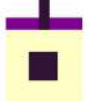


Elementos léxicos

```
while True:
        tries += 1
        user_number = int(input("
Enter a number to try to guess: "))

        if user_number < secret_number:
            print("Try with a higher number.")
        elif user_number > secret_number:
            print("Try with a lower number.")
        else:
            print(f"You won! It took you {tries}
tries")
            break
```

Convenciones

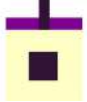
```
def main():
    """
    Input the upper and lower limit of a range of numbers
    and lets the user guess the random number until correct.
    """
    lower_limit = int(input("Enter the smaller number of the range: "))
    upper_limit = int(input("Enter the larger number of the range: "))
    secret_number = random.randint(lower_limit, upper_limit)
    tries = 0
```



A tomar en cuenta

- Sintaxis
- Literales
- Operadores
- import
- Condicionales
- Loops

```
"""
Guess a number game with the user.
"""

import random

def main():
    """Input the upper and lower limit of a range of numbers
    and lets the user guess the random number until correct.
    """
    lower_limit = int(input("Enter the smaller number of the range: "))
    upper_limit = int(input("Enter the larger number of the range: "))
    secret_number = random.randint(lower_limit, upper_limit)
    tries = 0

    while True:
        tries += 1
        user_number = int(input("Enter a number to try to guess: "))

        if user_number < secret_number:
            print("Try with a higher number.")
        elif user_number > secret_number:
            print("Try with a lower number.")
        else:
            print(f"You won! It took you {tries} tries")
            break

if __name__ == "__main__":
    main()
```



Estructuras propias

```
[]
# An empty list
["hello!!"]
# A list of one string
["hello!!", "oops"]
# A list of two strings
["hello!!", "oops", 10]
# A list of two strings and an int
["hello!!", ["oops", 10]]
# A list with a nested list
```

Operaciones esenciales

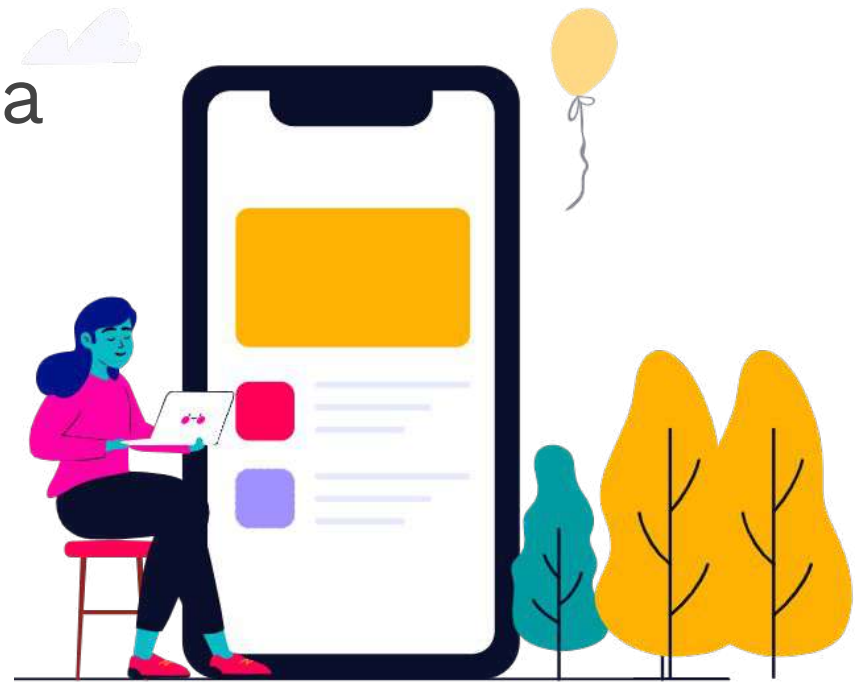
Estructuras propias:

- Listas
- Tuplas
- Conjuntos (sets)
- Diccionarios

```
items_list = []
# items_list is []
items_list.append(34)
# items_list is [34]
items_list.append(22)
# items_list is [34, 22]
items_list.sort()
# items_list is [22, 34]
items_list.pop()
# Returns 34; items_list is [22]
items_list.insert(1, 34)
# items_list is [22, 34]
items_list.insert(1, 55)
# items_list is [22, 55, 34]
items_list.pop(1)
# Returns 55; items_list is [22, 34]
items_list.remove(22)
# items_list is [34]
items_list.remove(55)
# raises ValueError
```

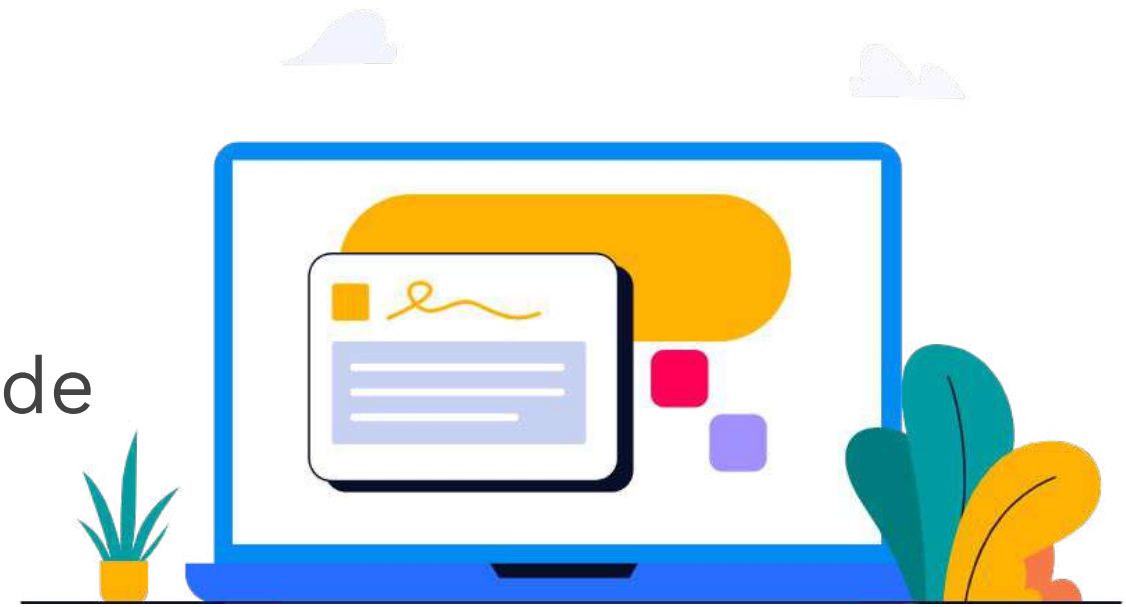
Funciones

- Declaración y llamada
- Recursivas
- Anidadas
- High order functions
- Lambdas



Además

- Manejo de excepciones y errores.
- Manipulación de archivos.



¿Estás lista?

Tipos de colecciones

¿Qué es una colección?

“Grupo de cero o más elementos que pueden tratarse como una unidad conceptual”.

¿Cero o más elementos?

Non-zero value

0

null

undefined

Tipos de colecciones

- Dinámicas.
- Inmutables.

Lineales

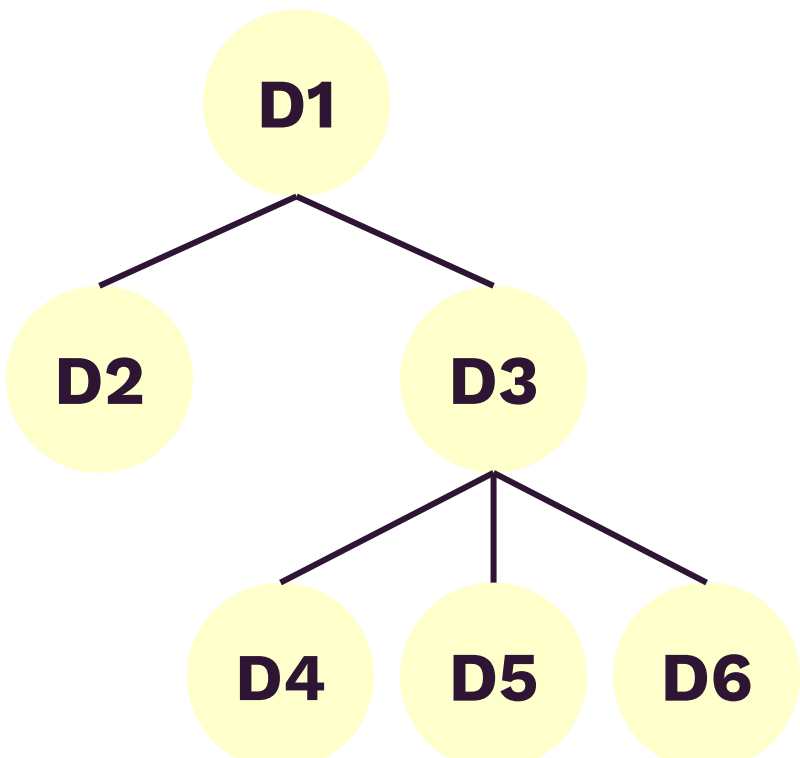
- Ordenadas por posición.
- Solo el primer elemento NO tiene predecesor.

Lineales

- Fila en el supermercado.
- Pila de platos.
- Checklist.

Jerárquicas

- Ordenadas como un árbol invertido.
- Solo el primer elemento NO tiene predecesor.
- Padres e hijos.

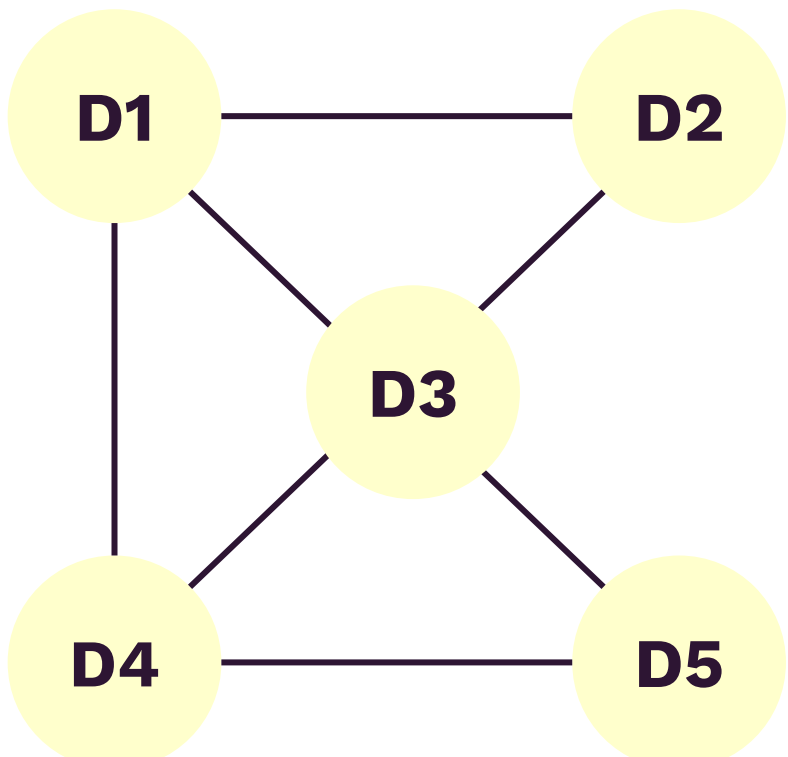


Jerárquicas

- Sistema de directorios.
- Índices de libros.

Grafos

- Cada dato puede tener varios predecesores y sucesores.
- Estos se llaman “vecinos”.

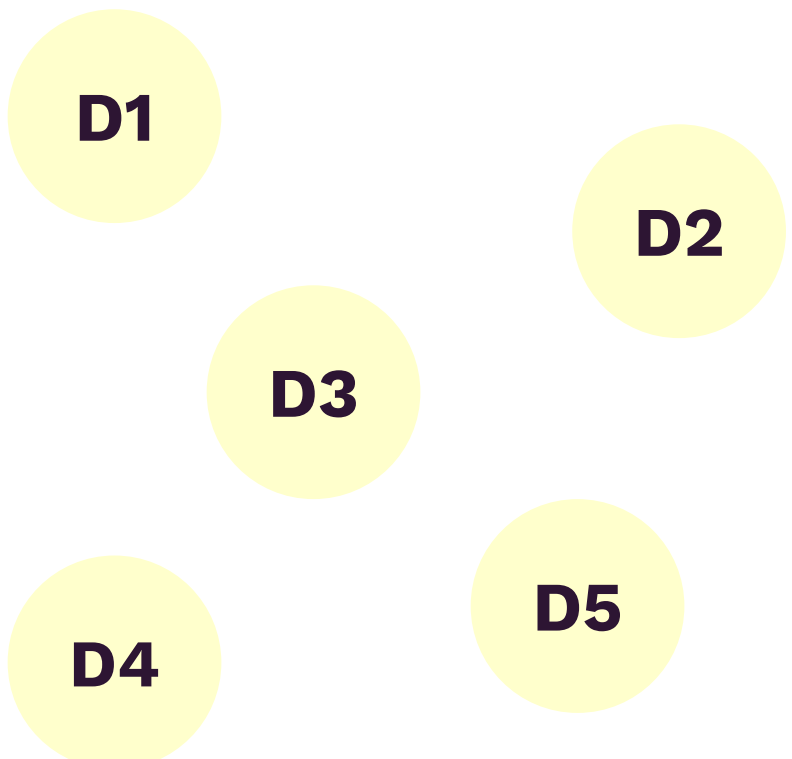


Grafos


ICN
HND
NRT
HKG
TPE
KUL
SIN
WAR
SYD
AKL
HNL
PPT
SEA
YYC
SFO
LAX
ORD
DAL
HOU
DCA
IAD
FLL
MEX
LCY
LGW
LHR
FRA
ARN
HAM
MAD
IST
TLV
DOH
DXB
AUH

Desordenadas

- No tienen orden particular.
- No hay predecesores o sucesores.



Desordenadas

Ordenadas

- Impone un orden con una regla.

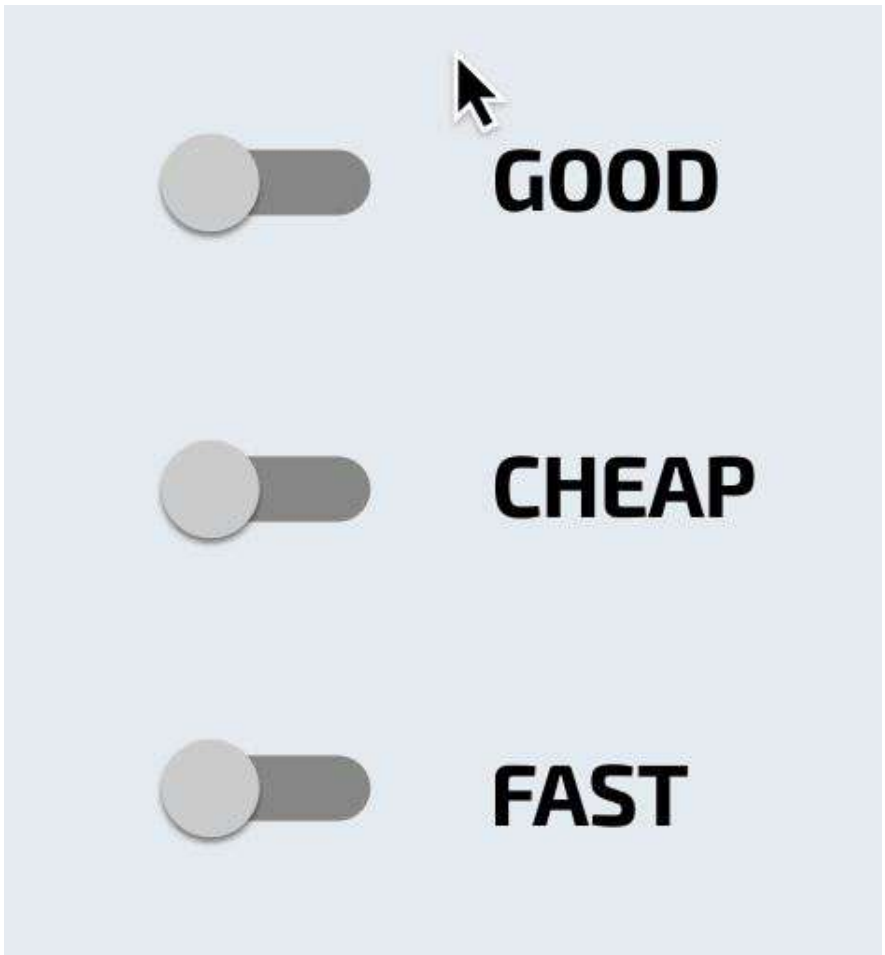
  $item_i <= item_{i+1}$,

Taxonomía

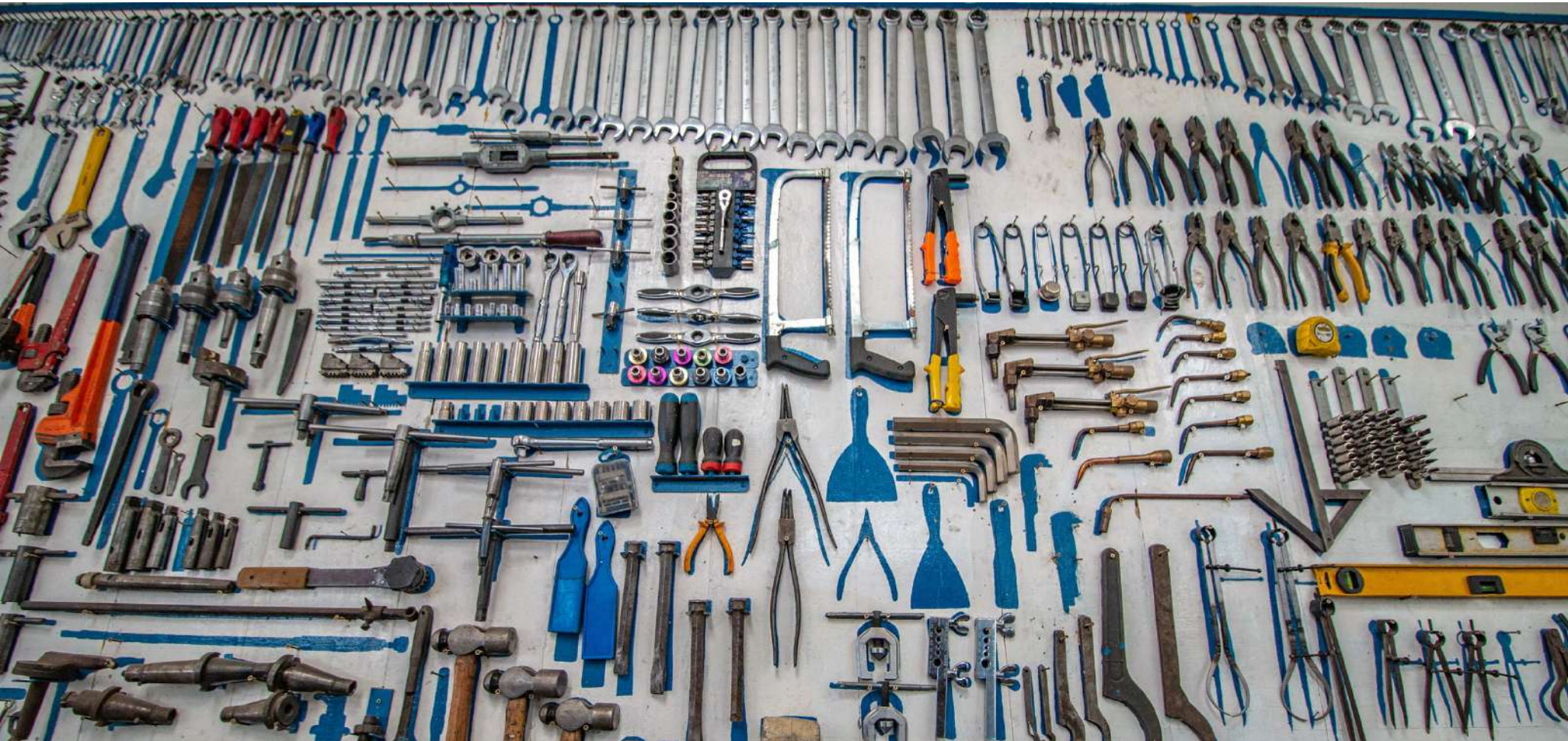
- **Colección**
  - **Grafos**
  - **Jerárquicas**
    - Árbol binario
    - Montículo
  - **Lineales**
    - Lista
      - Lista ordenada
    - Cola
      - Cola de prioridad
    - Pila
    - String
  - **Desordenadas**
    - Bolsa
      - Bolsa ordenada
    - Diccionario
      - Diccionario ordenado
    - Conjunto
      - Conjunto ordenado

Supongamos que...

El eterno dilema...



¡Son herramientas!



Operaciones esenciales en colecciones

Operaciones

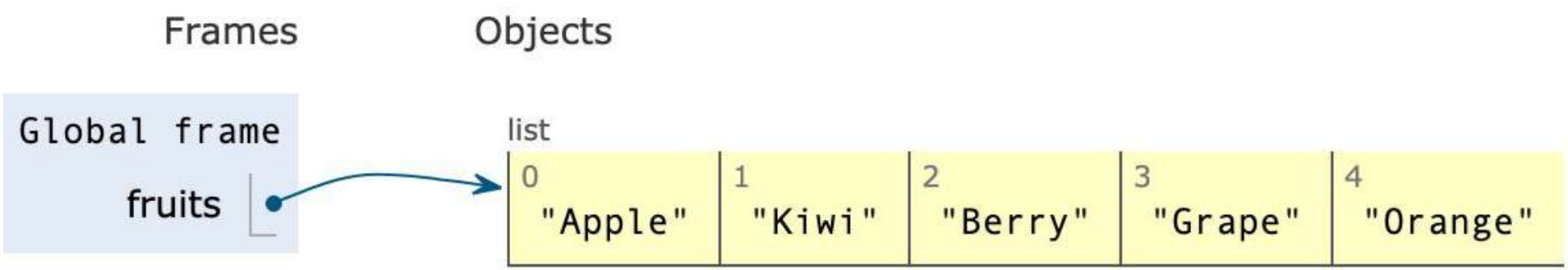
- Tamaño
- Pertenencia
- Recorrido
- String
- Igualdad

Operaciones

- Concatenación
- Conversión de tipo
- Insertar
- Remover
- Reemplazar
- Acceder

🐍 Creando una lista

```
fruits = ["Apple", "Kiwi", "Berry", "Grape", "Orange"]
```



Colecciones incorporadas en Python

Listas/Lists

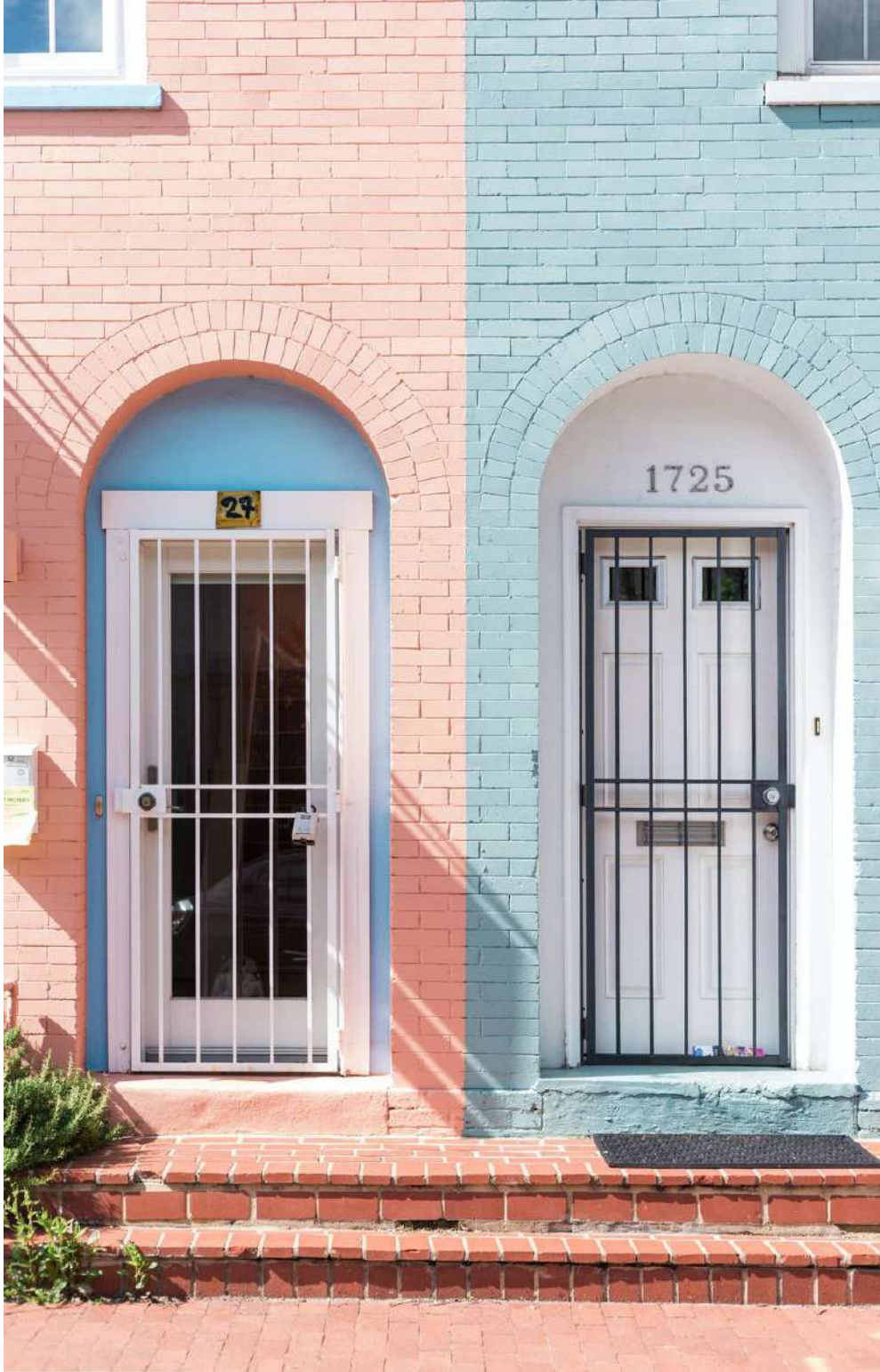
- Propósito general
- Estructura más utilizada
- Tamaño dinámico
- De tipo secuencial
- Ordenable

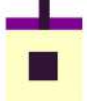
Listas/Lists

```
list1 = list()
list2 = ['t', 25, 'cat', 3.1415]
list3 = [number**2 for number in range(1, 100, 3)]

print(list1)
print(list2)
print(list3)

>>> []
>>> ['t', 25, 'cat', 3.1415]
>>> [1, 16, 49, 100, 169, 256, 361, 484, 625, 784, 961, 1156,
1369, 1600, 1849, 2116, 2401, 2704, 3025, 3364, 3721, 4096,
4489, 4900, 5329, 5776, 6241, 6724, 7225, 7744, 8281, 8836,
9409]
```

Tuplas/Tuples

- Inmutable (no se pueden añadir o cambiar).
- Útiles para datos constantes.
- Más rápidas que las listas.
- Tipo secuencial.



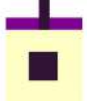


Tuplas/Tuples

```
tuple1 = ()
tuple2 = (1274, 1275, 1276)
tuple3 = 'mulan', 'pucca', 'percy',

print(tuple1)
print(tuple2)
print(tuple3)

>>> ()
>>> (1274, 1275, 1276)
>>> ('mulan', 'pucca', 'percy')
```

Conjuntos /Sets

- Almacenan objetos no-duplicados.
- De acceso rápido.
- Aceptan operaciones lógicas.
- Son desordenados.

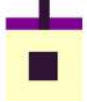


Conjuntos/Sets

```
set1 = {3, 5, 9, 3, 9}
set2 = set()
numbers = [1, 2, 3, 4, 5, 6, 1, 2]
set3 = set(numbers)

print(set1)
print(set2)
print(set3)

>>> {9, 3, 5}
>>> set()
>>> {1, 2, 3, 4, 5, 6}
```

Diccionarios/ Dictionaries

- Pares de llave/valor.
- Arrays asociativos (hash maps).
- Son desordenados.



Diccionarios/Dictionaries

```
cats1 = {
    'mulan': 2,
    'pucca': 1.2,
    'percy': 4,
}
cats2 = dict([('mulan', 2),('pucca', 1.2), ('percy', 4)])
cats3 = dict(mulan=2, pucca=1.2, percy=4)

print(cats1)
print(cats2)
print(cats3)

>>> {'mulan': 2, 'pucca': 1.2, 'percy': 4}
>>> {'mulan': 2, 'pucca': 1.2, 'percy': 4}
>>> {'mulan': 2, 'pucca': 1.2, 'percy': 4}
```

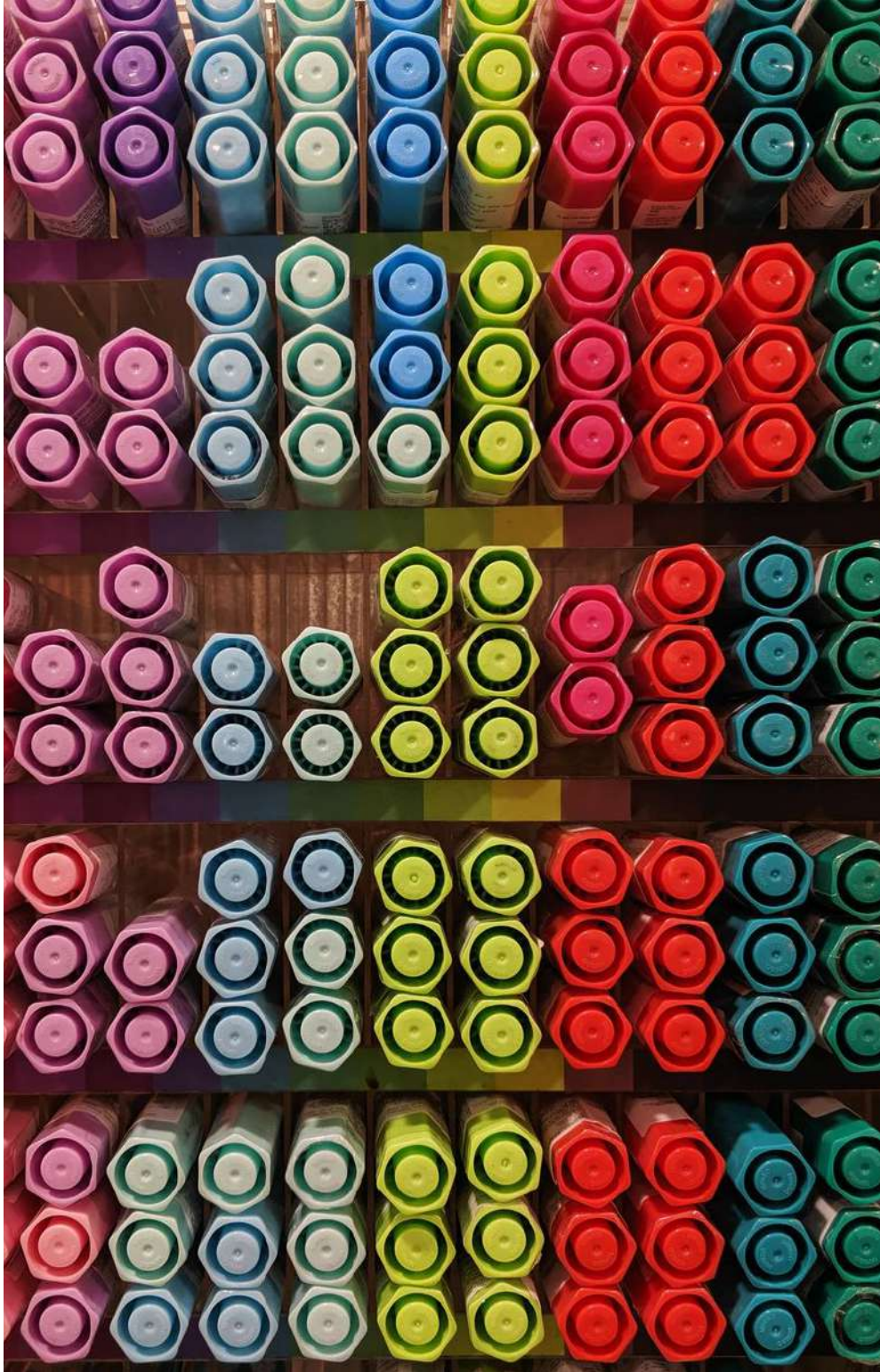
Reto

- ¿En qué casos usarías cada estructura?
- ¿Qué tipo de información guardarías?

Arrays

¿Qué es una estructura de datos?

Representación interna de una colección de información.

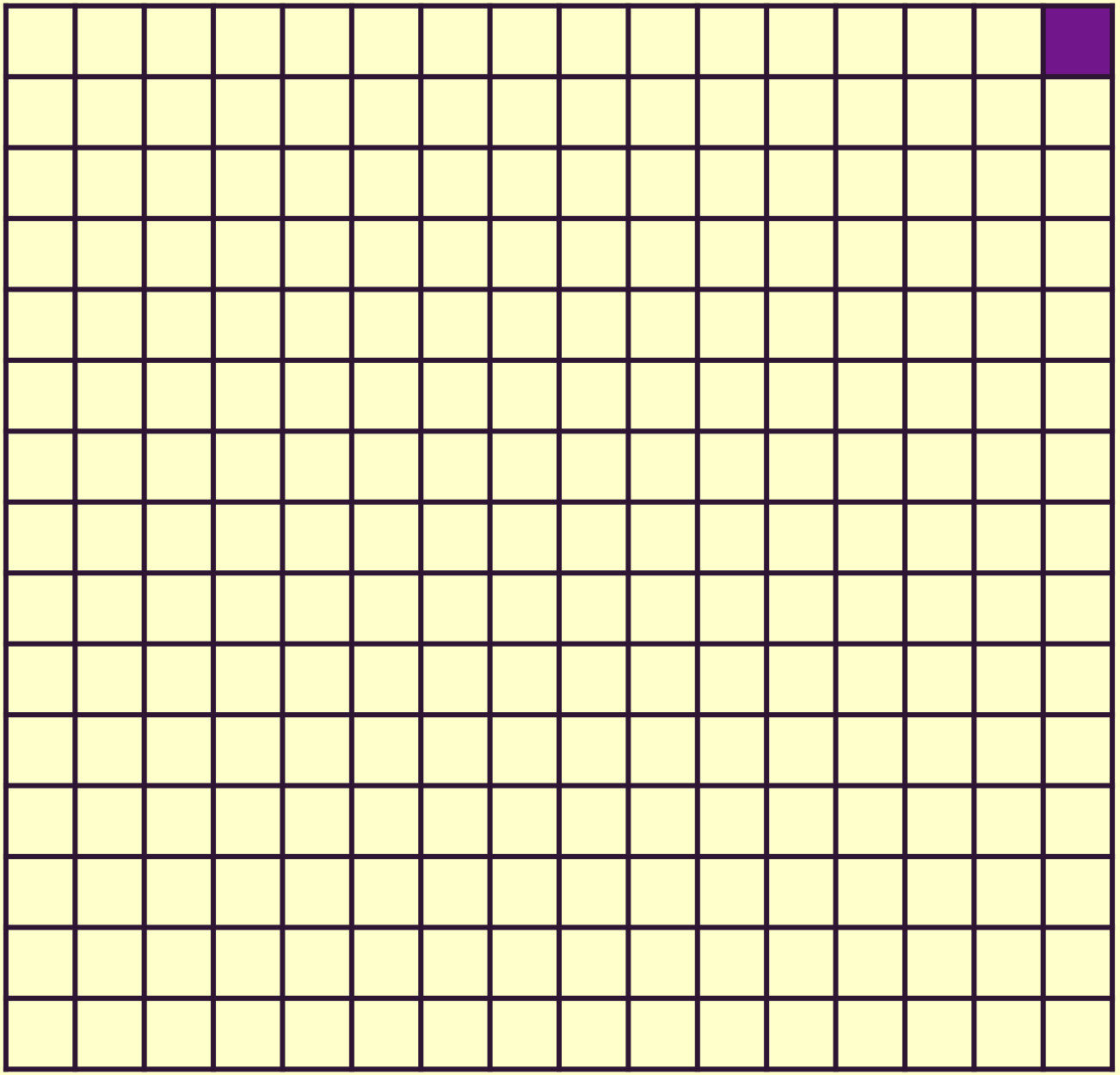


Conceptos clave

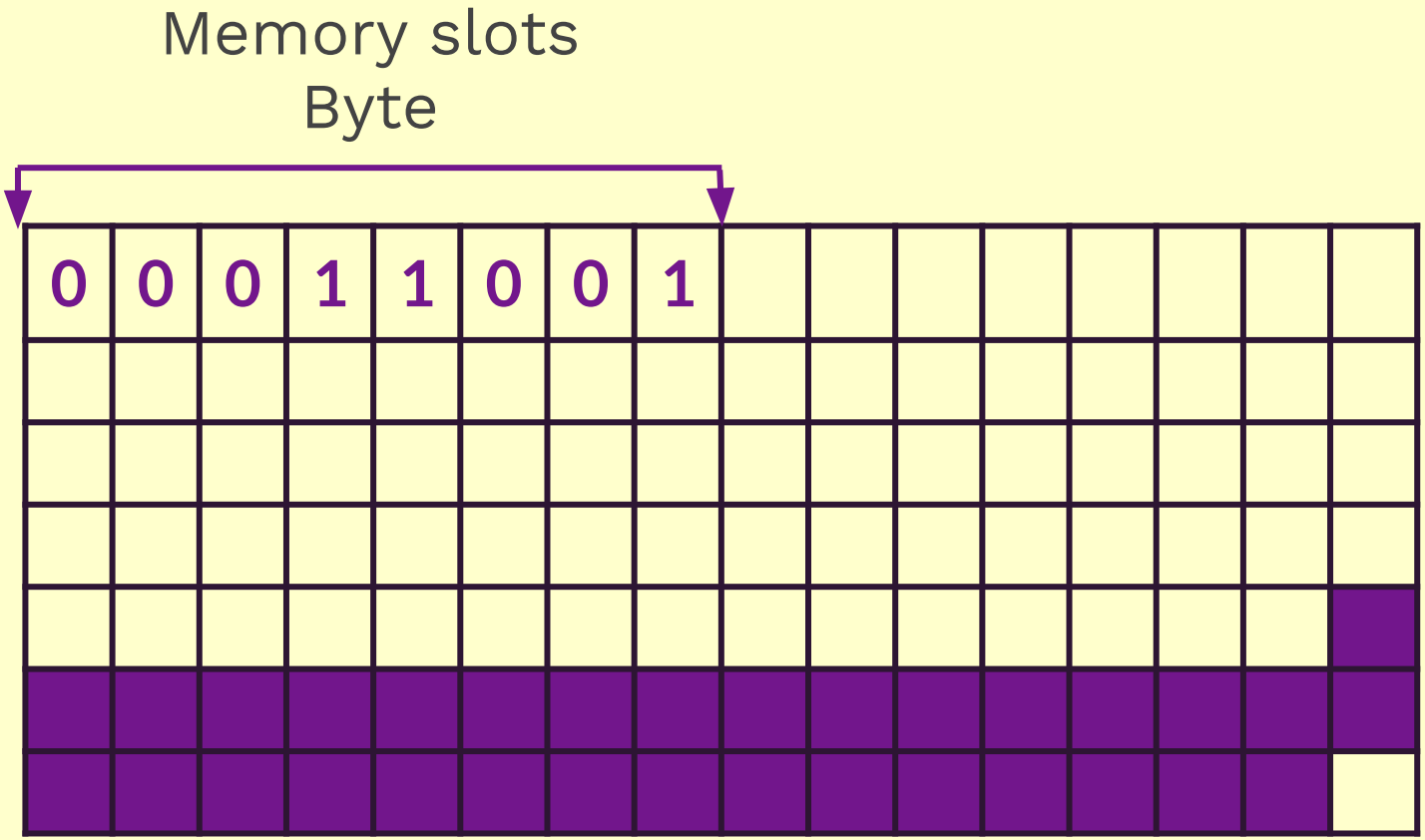
- **Elemento:** valor almacenado en las posiciones del array.
- **Índice:** referencia a la posición del elemento.

Memoria

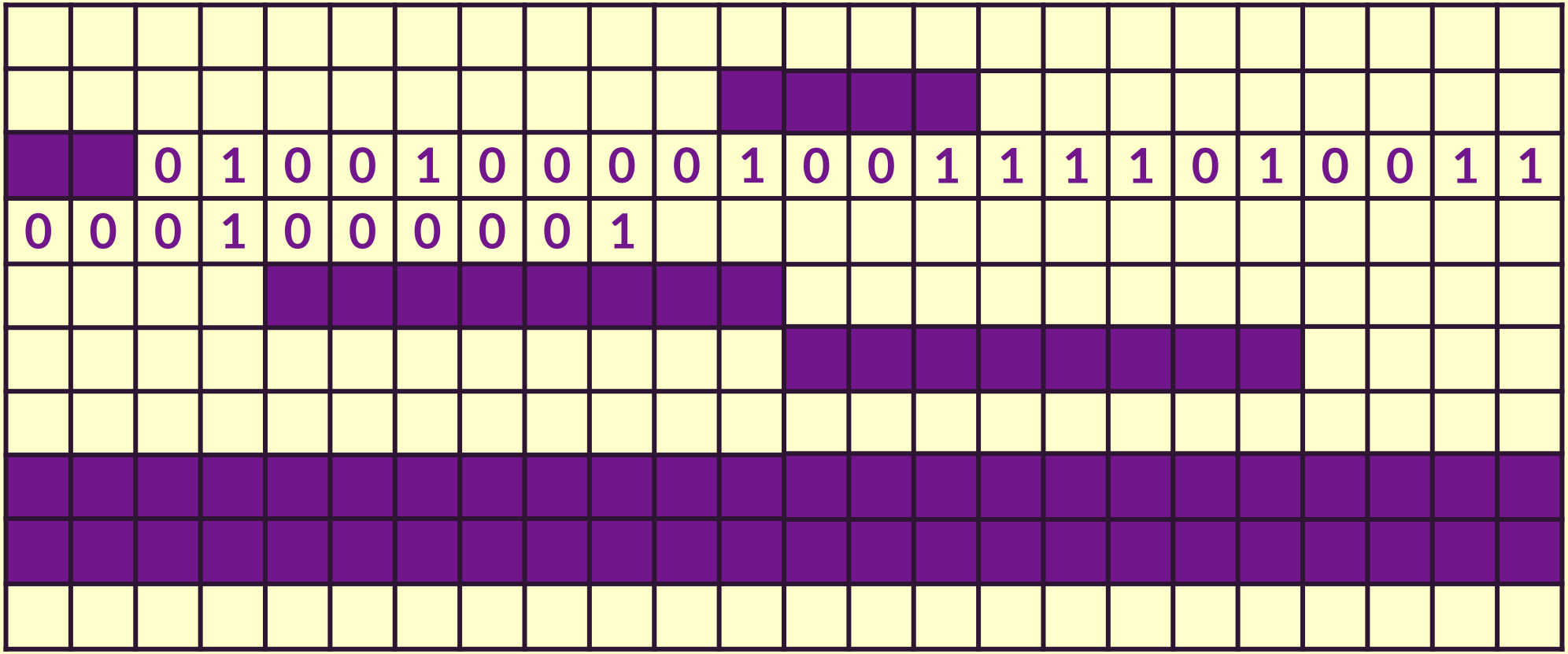
Memory Slot



¿Cómo se guardan los datos?

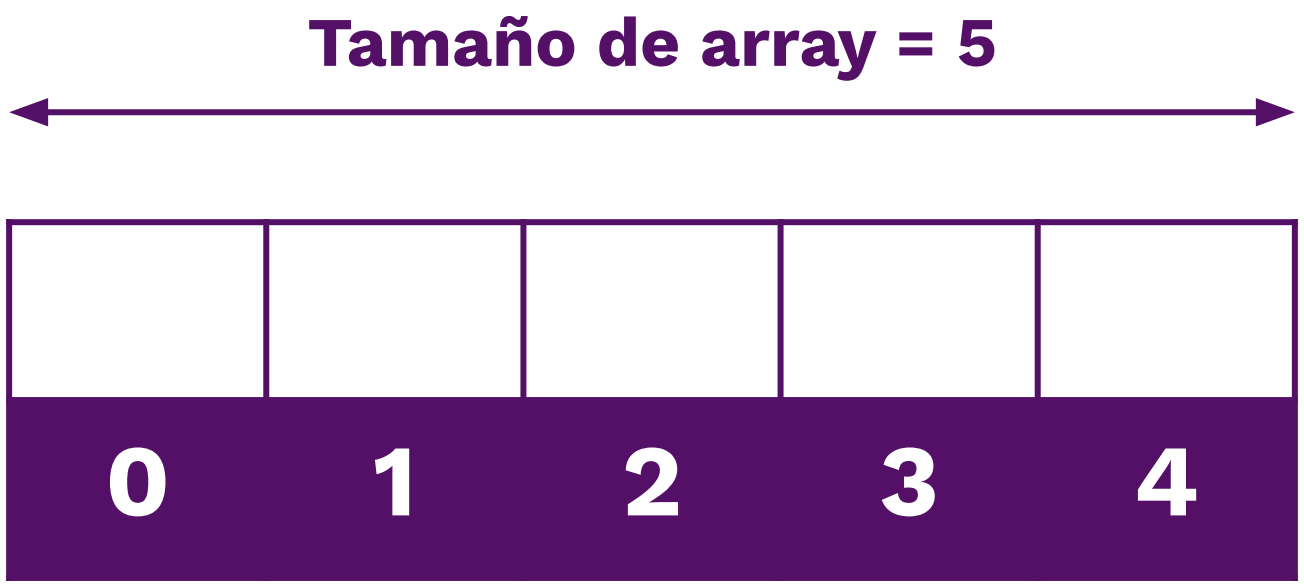


EDAD = 25



SALUDO = “Hola”

Array



Array

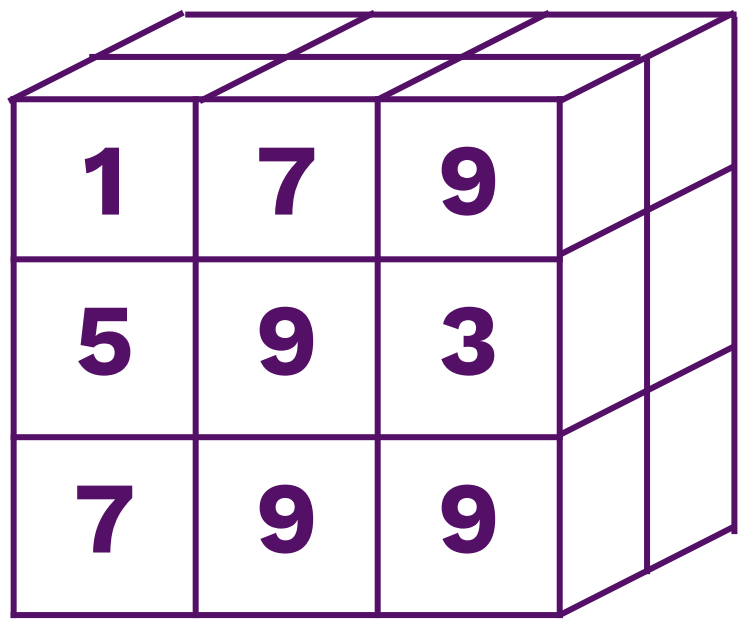
1D Array

3	2

2D Array

1	0	1
3	4	1

3D Array



¿Los array son listas?

Los array son restrictivos

No pueden:

- Agregar posiciones.
- Remover posiciones.
- Modificar su tamaño.
- Su capacidad se define al crearse.

¿Dónde se utilizan?

- Un videojuego con sprites de 100x100 px.
- Opciones en un menú.

Módulo ‘array’

- Solo almacena números y caracteres.
- Basado en listas.

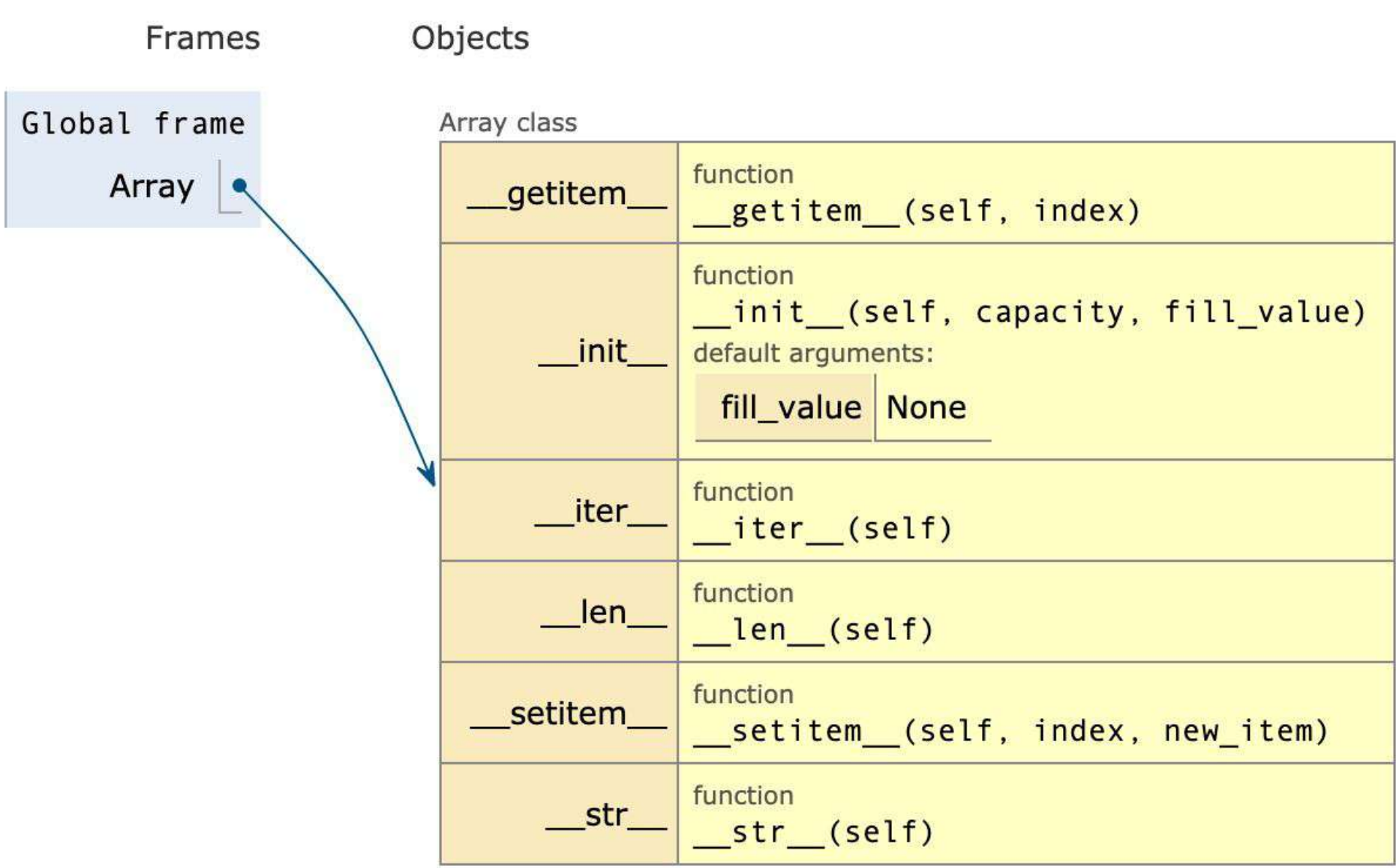
Crear un array

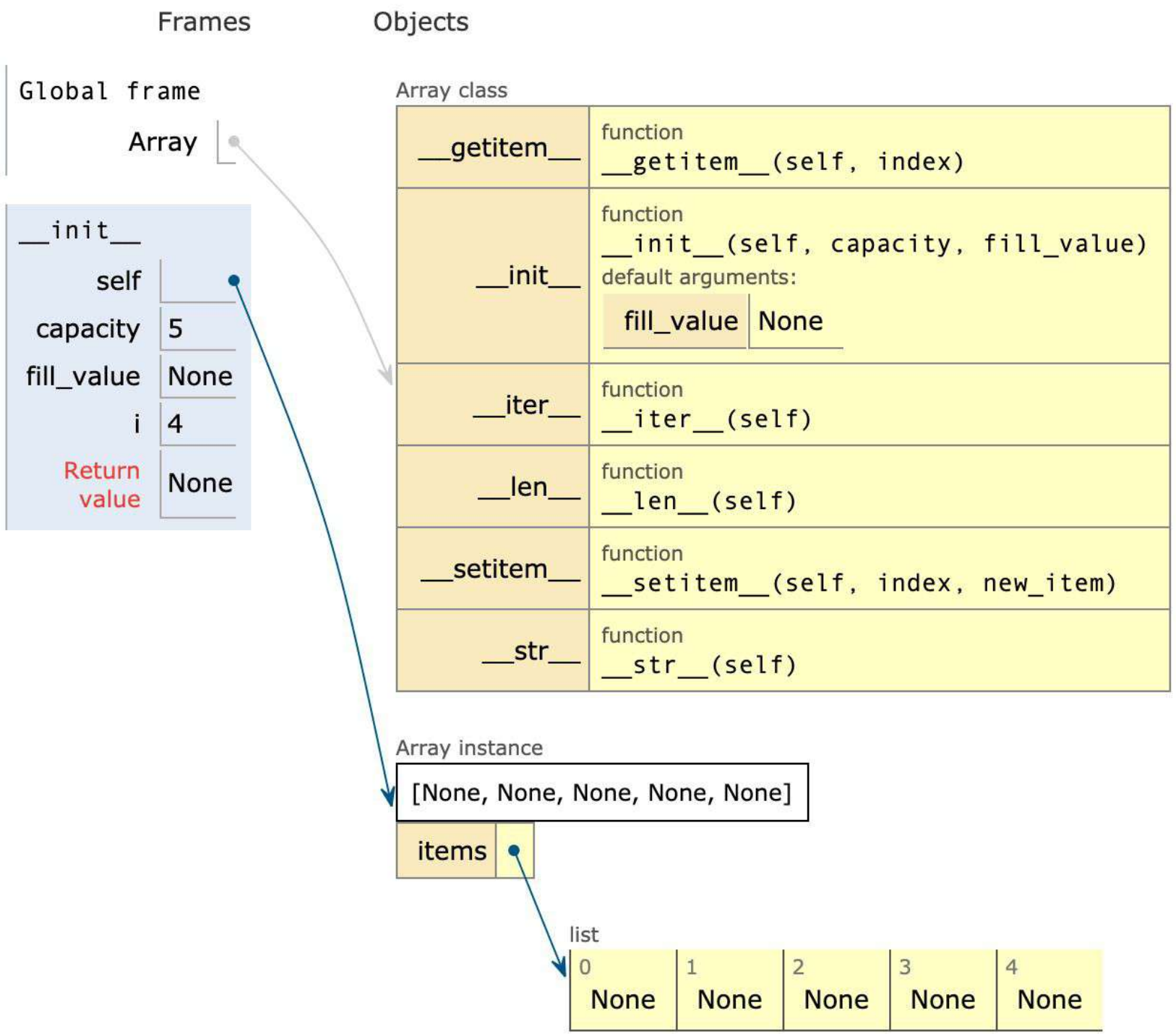
🐍 Nuestro propio array

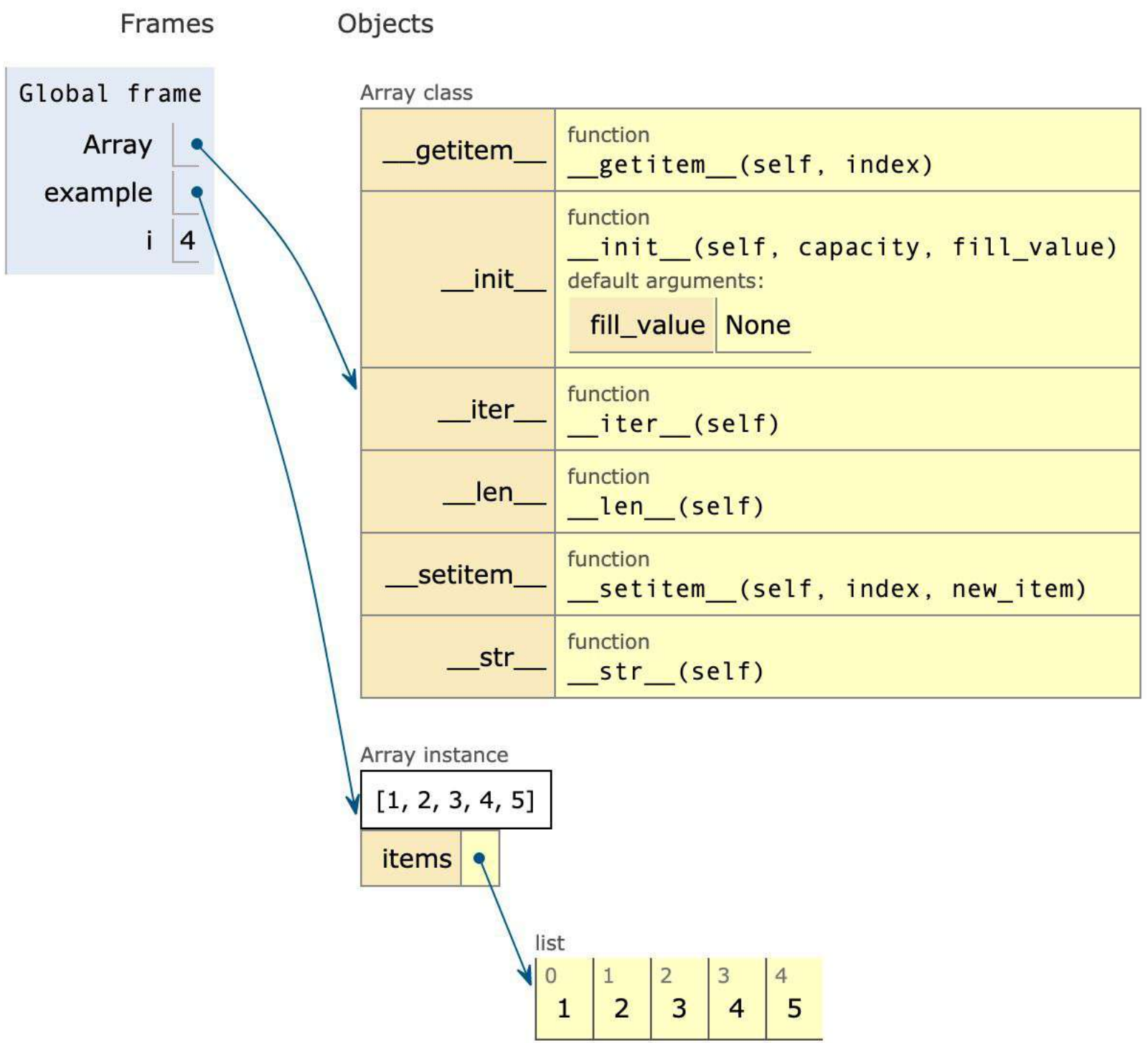
Métodos:

- Crearse.
- Longitud.
- Representación string.
- Pertenencia.
- Índice.
- Reemplazo.

¿Qué hicimos?


Frames
Objects
Global frame
Array
Array class
__getitem__
function
__getitem__(self, index)
__init__
function
__init__(self, capacity, fill_value)
default arguments:
fill_value
None
__iter__
function
__iter__(self)
__len__
function
__len__(self)
__setitem__
function
__setitem__(self, index, new_item)
__str__
function
__str__(self)


Frames
Objects
Global frame
Array
__init__
self
capacity
5
fill_value
None
i
4
Return value
None
Array class
__getitem__
function
__getitem__(self, index)
__init__
function
__init__(self, capacity, fill_value)
default arguments:
fill_value
None
__iter__
function
__iter__(self)
__len__
function
__len__(self)
__setitem__
function
__setitem__(self, index, new_item)
__str__
function
__str__(self)
Array instance
[None, None, None, None, None]
items
list
0
None
1
None
2
None
3
None
4
None


Frames
Objects
Global frame
Array
example
i
4
Array class
__getitem__
function
__getitem__(self, index)
__init__
function
__init__(self, capacity, fill_value)
default arguments:
fill_value
None
__iter__
function
__iter__(self)
__len__
function
__len__(self)
__setitem__
function
__setitem__(self, index, new_item)
__str__
function
__str__(self)
Array instance
[1, 2, 3, 4, 5]
items
list
0
1
1
2
2
3
3
4
4
5

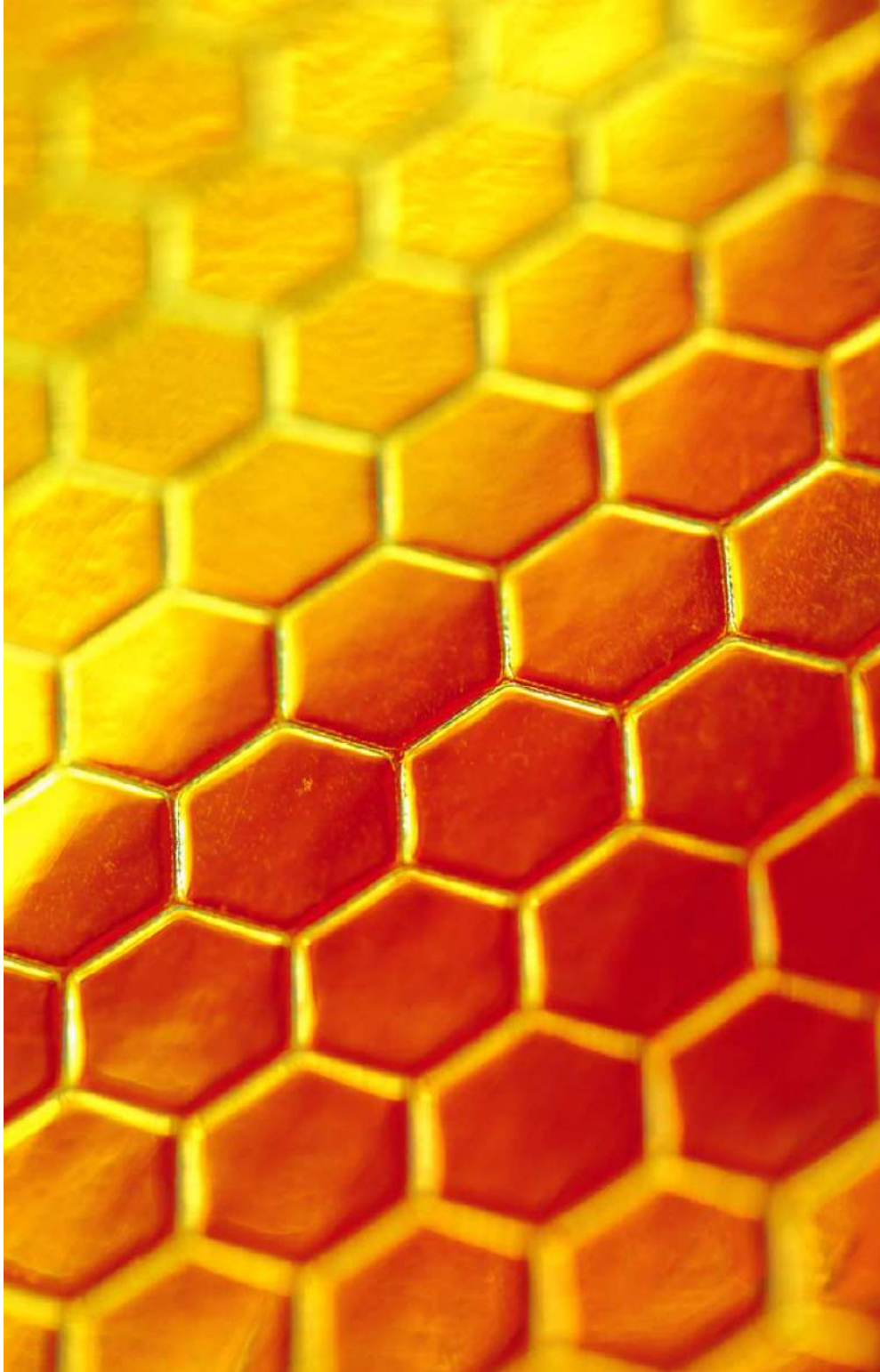
Reto

1. Crea una clase de array.
2. Incorpora un método para poblar sus slots con números aleatorios o secuenciales.
3. Incluye un método que sume todos los valores del array.

Crear un array de dos dimensiones

Otros nombres

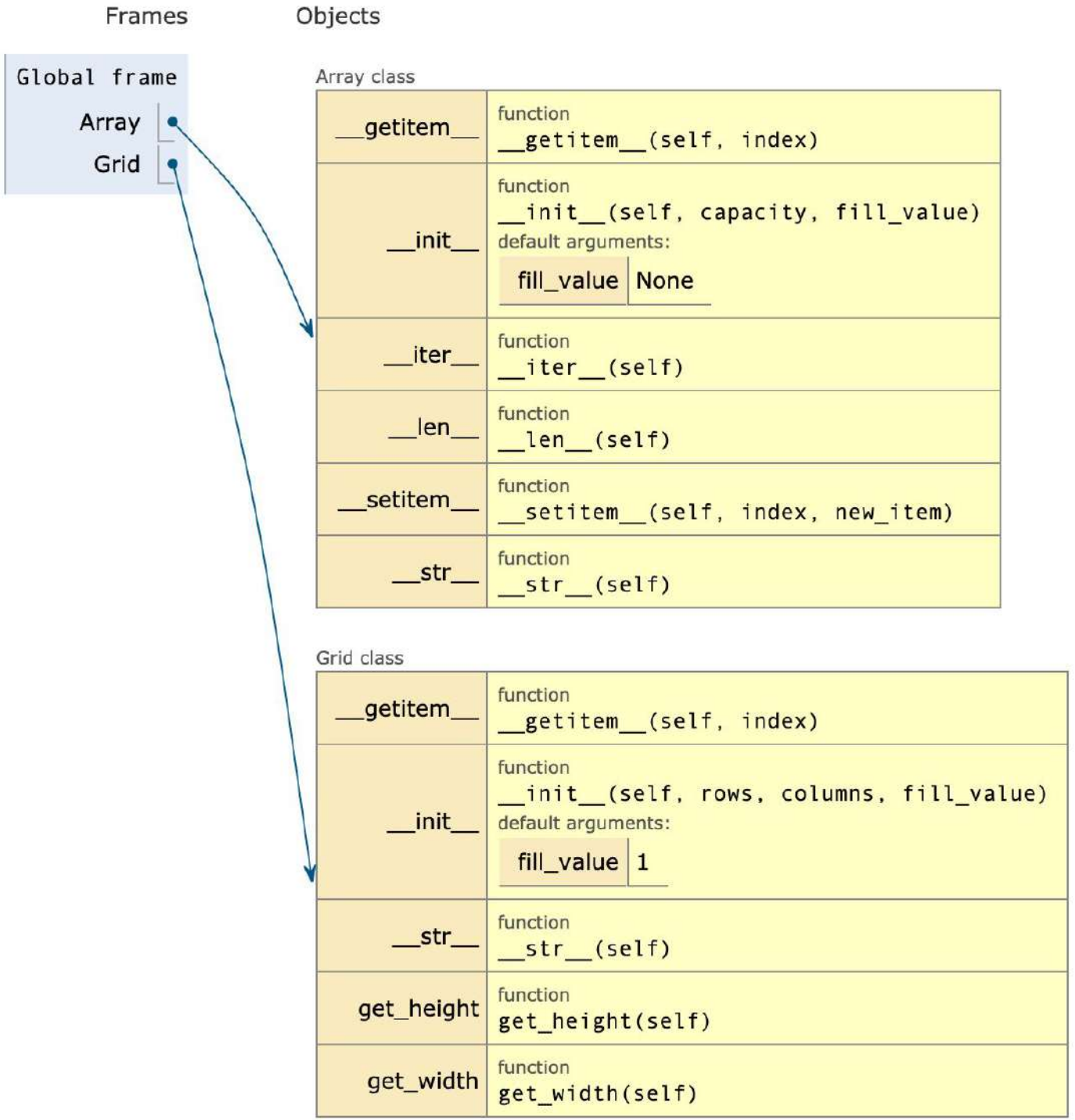
- Bi-dimensional array
- Two-dimensional array
- Grid
- Rejilla
- Malla
- Tabla

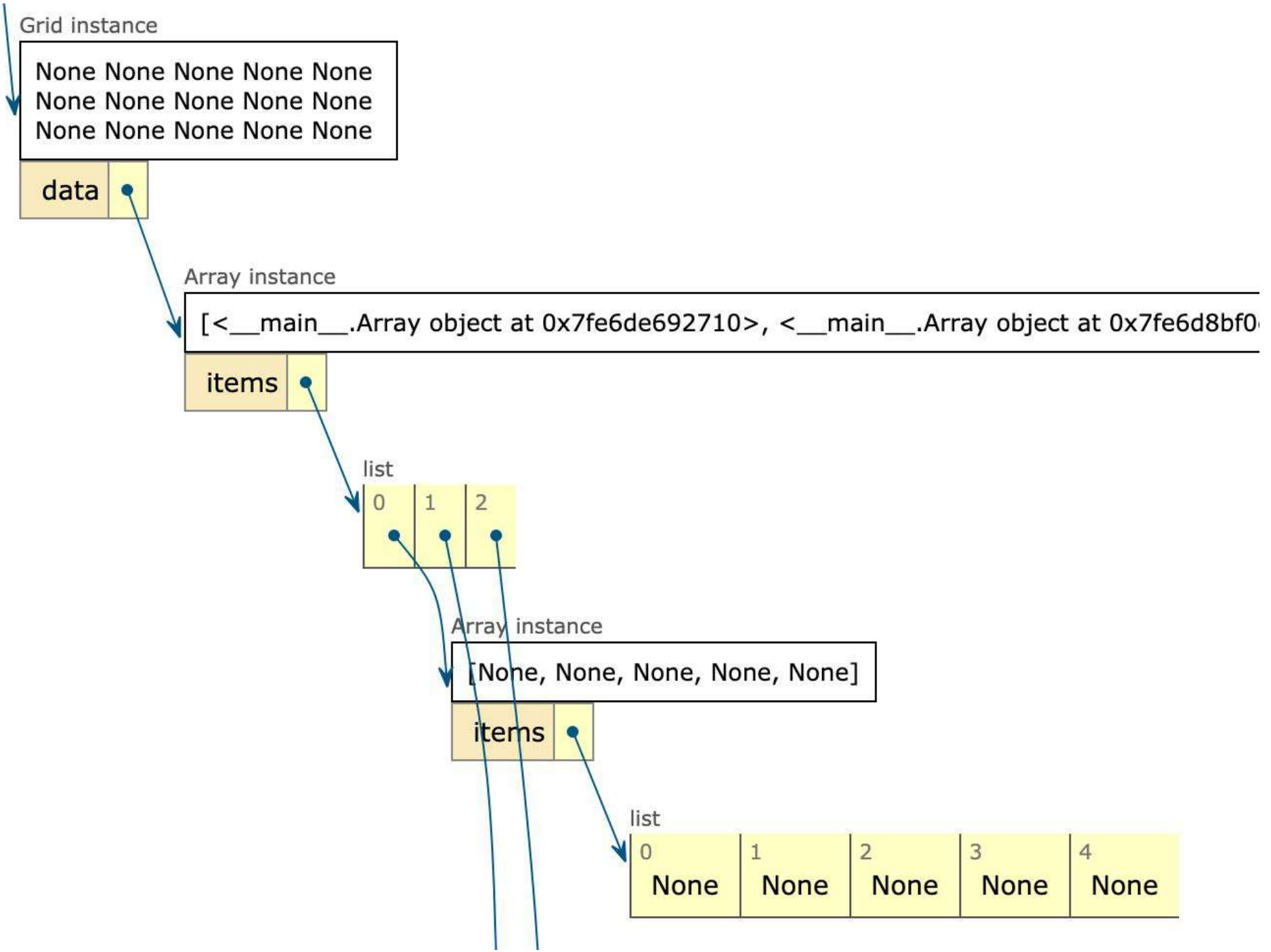


Estructura

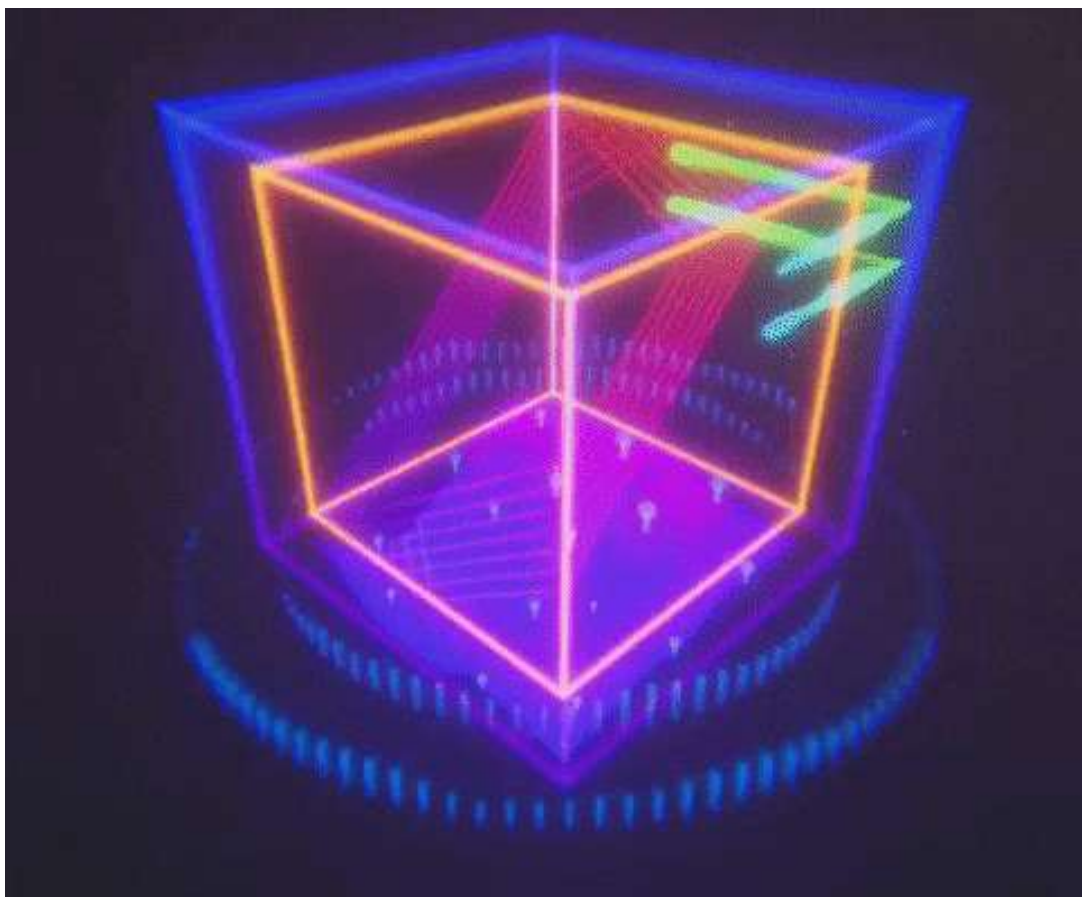
	Columna 0	Columna 1	Columna 2
Fila 0	0	1	2
Fila 1	3	4	5
Fila 2	6	7	8

¿Qué hicimos?


Frames
Objects
Global frame
Array
Grid
Array class
__getitem__
function
__getitem__(self, index)
__init__
function
__init__(self, capacity, fill_value)
default arguments:
fill_value
None
__iter__
function
__iter__(self)
__len__
function
__len__(self)
__setitem__
function
__setitem__(self, index, new_item)
__str__
function
__str__(self)
Grid class
__getitem__
function
__getitem__(self, index)
__init__
function
__init__(self, rows, columns, fill_value)
default arguments:
fill_value
1
__str__
function
__str__(self)
get_height
function
get_height(self)
get_width
function
get_width(self)


Grid instance
None None None None None
None None None None None
None None None None None
data
Array instance
[<__main__.Array object at 0x7fe6de692710>, <__main__.Array object at 0x7fe6d8bf0
items
list
0
1
2
Array instance
[None, None, None, None, None]
items
list
0
None
1
None
2
None
3
None
4
None

¿Y los de tres dimensiones?



Reto

1. Crea una clase ‘Grid’.
2. Incorpora un método para poblar sus slots.
3. Crea una clase ‘Cube’: array de 3 dimensiones.

Nodos y singly linked list

Linked structures

- Consiste de nodos conectados a otros.
- Los más comunes son *sencillos* o *dobles*.
- No se accede por índice, sino por recorrido.

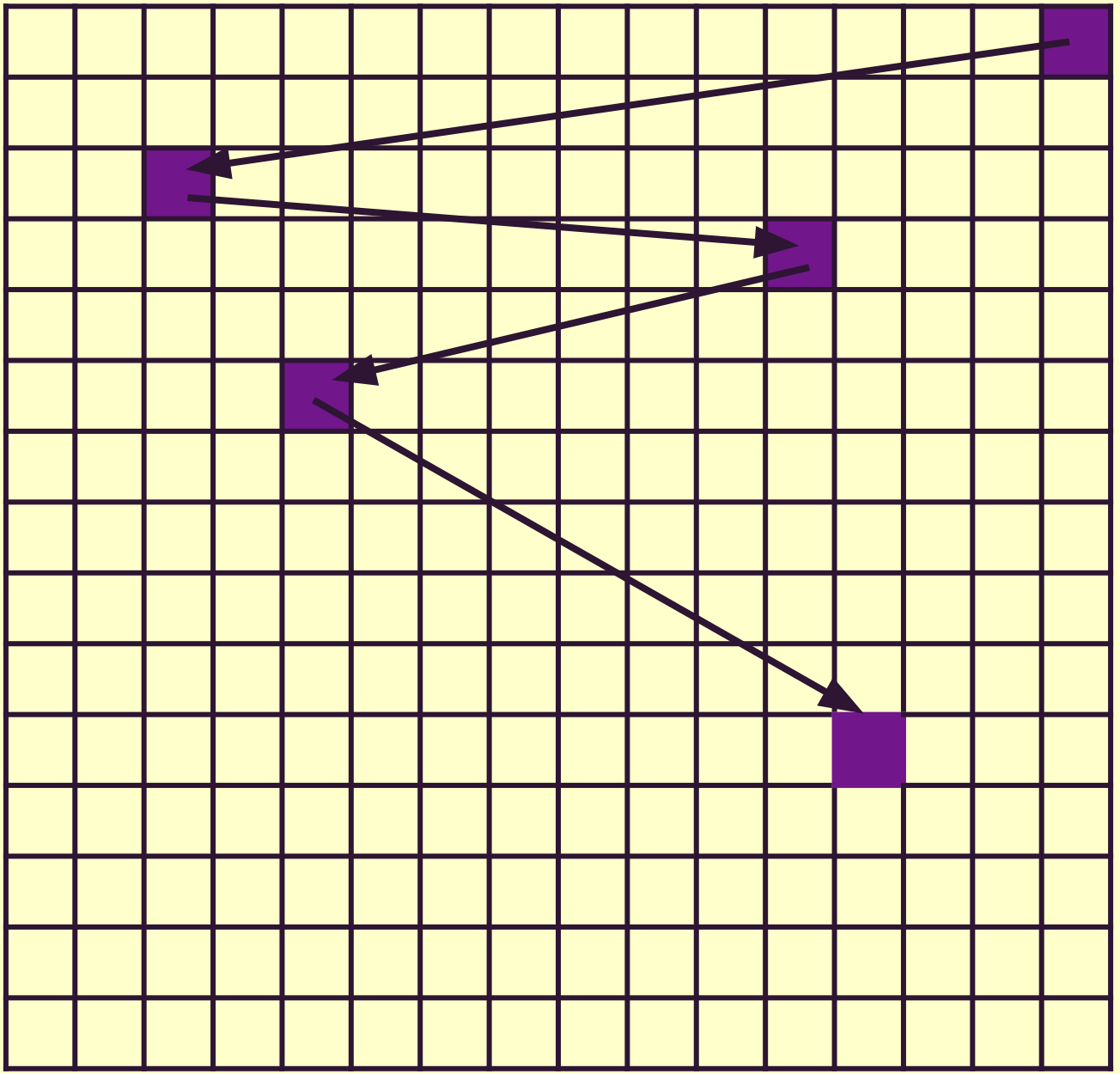
Conceptos clave

- **Data:** valor almacenado en nodos.
- **Next:** referencia al siguiente nodo.
- **Previous:** referencia al nodo anterior.

Conceptos clave

- **Head:** referencia al primer nodo.
- **Tail:** referencia al último nodo.

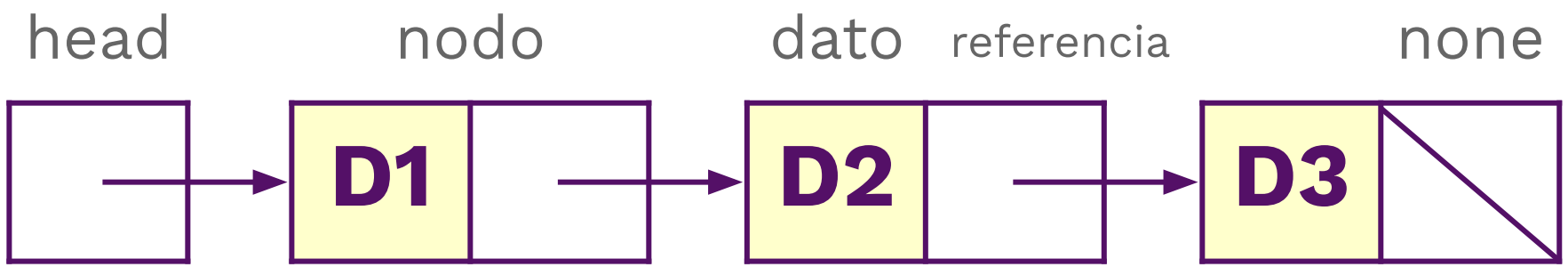
¿Cómo se guardan las LinkedList?



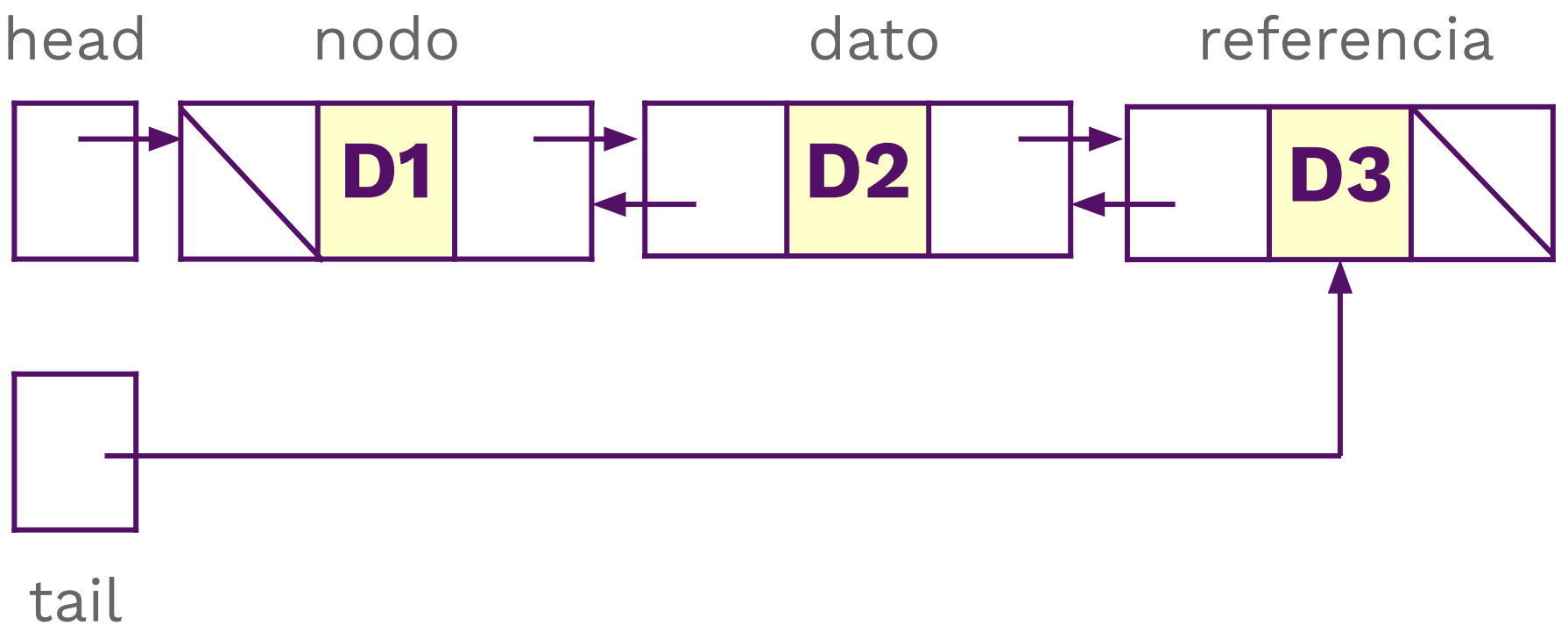
Implementación

- Implementar otras estructuras.
- Optimización.

Singly Linked Structure



Double Linked Structure



Linked lists

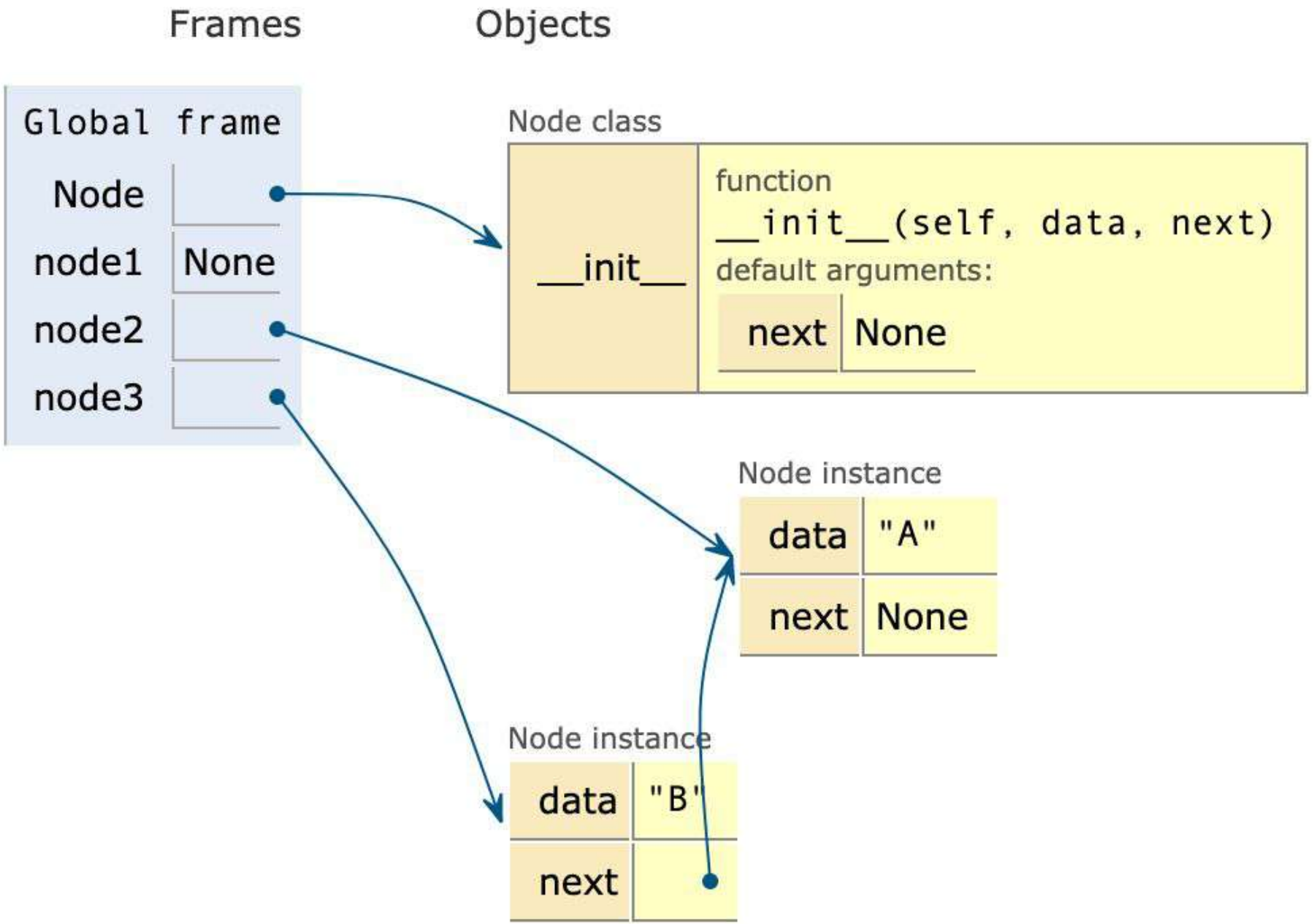
- Hacer/rehacer operaciones en un editor de texto.
- Historial de un navegador.

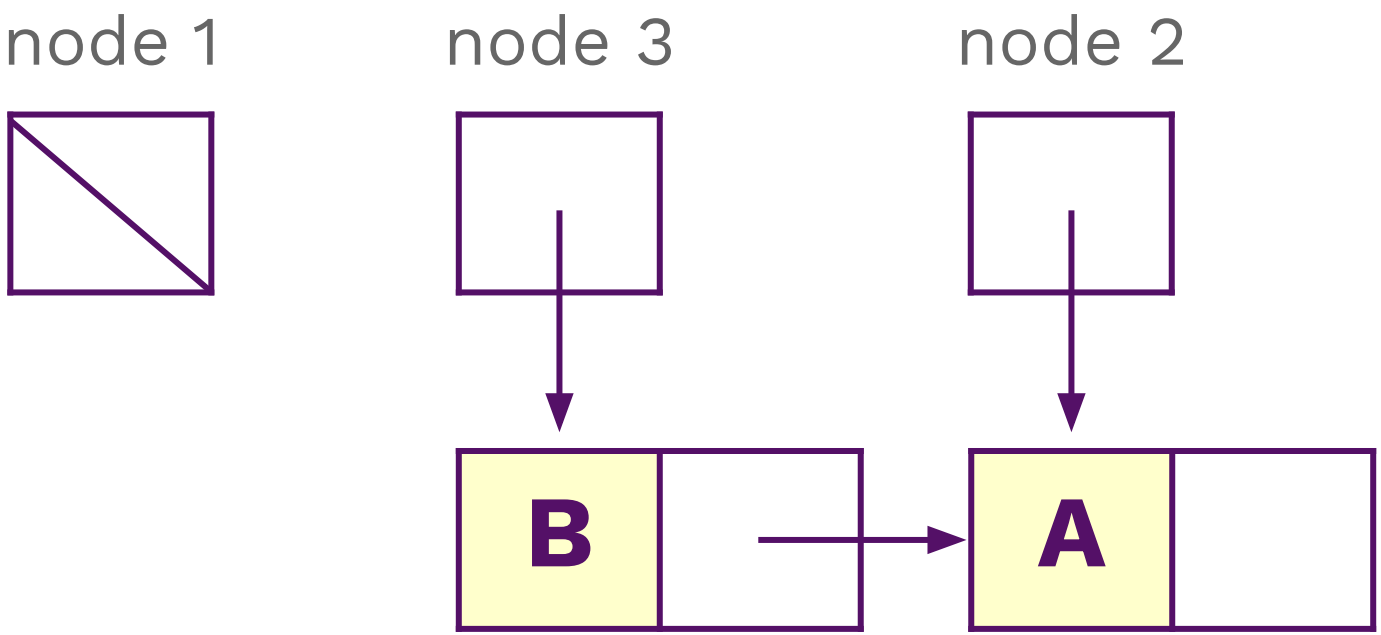
Crear nodos

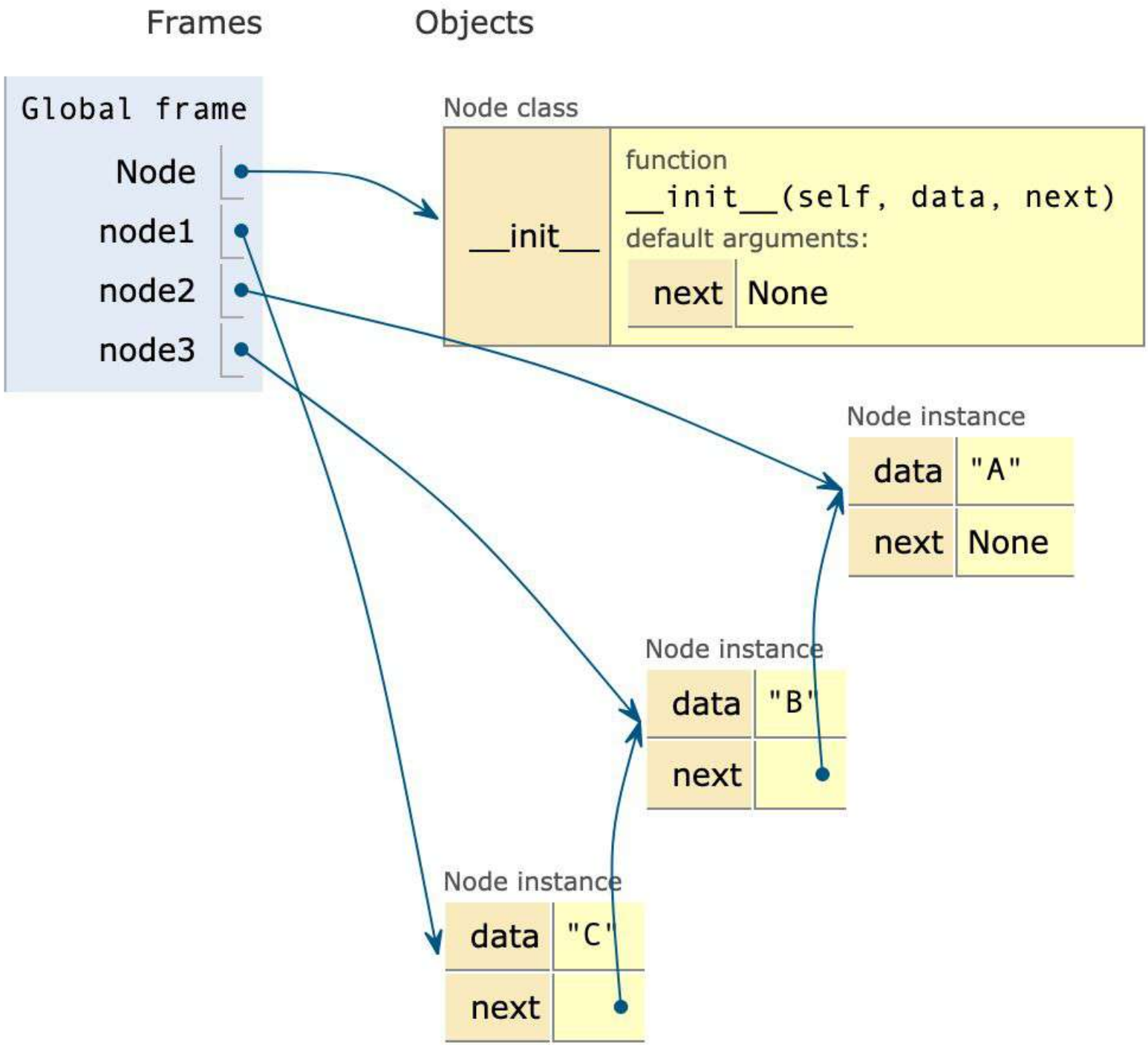
🐍 Singly linked list con nodos

- Creamos una clase *Node*.
- Referimos valores.
- Unimos nodos iterando.

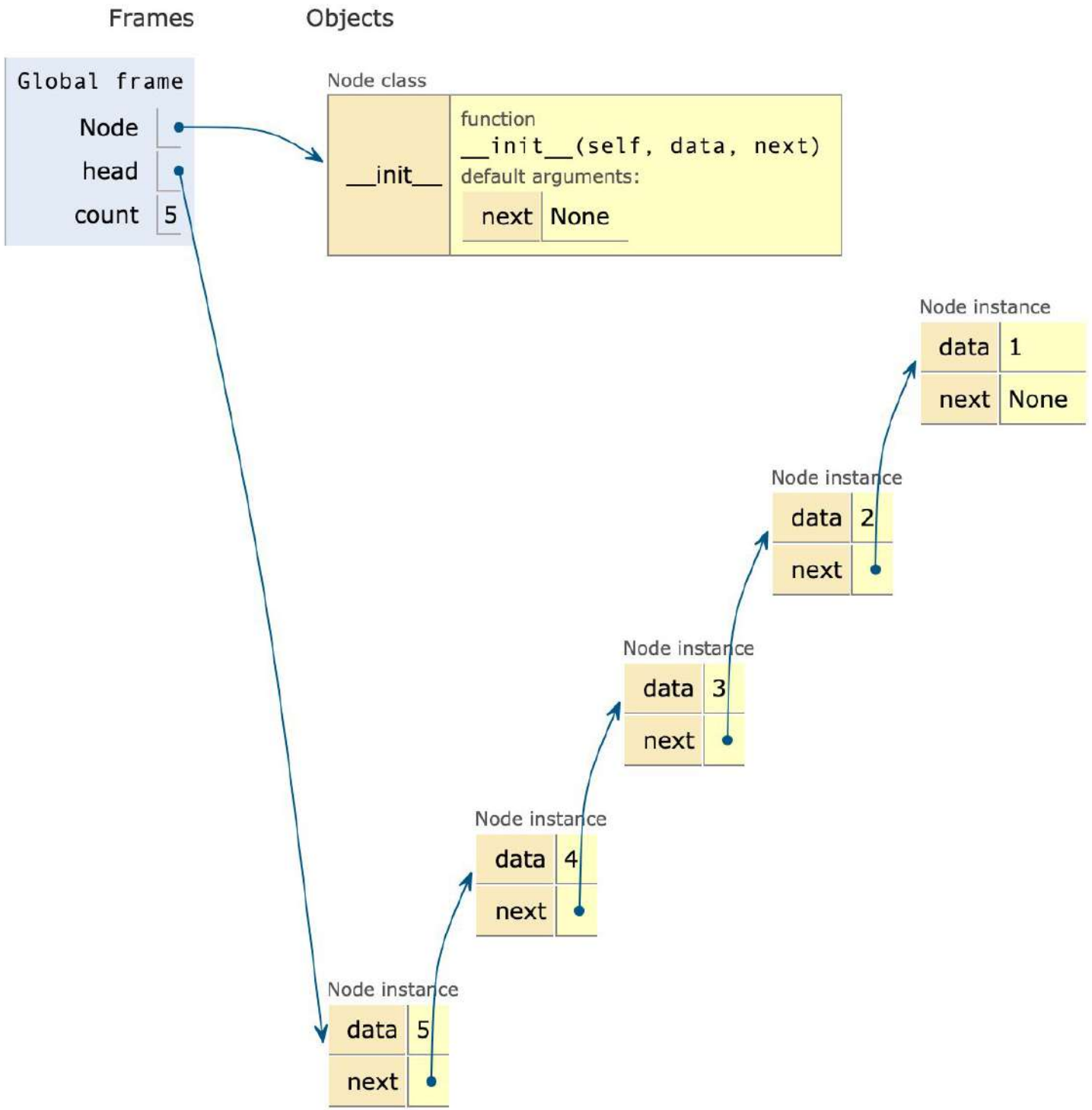
¿Qué hicimos?


Frames
Objects
Global frame
Node
node1
None
node2
node3
Node class
__init__
function
__init__(self, data, next)
default arguments:
next
None
Node instance
data
"A"
next
None
Node instance
data
"B"
next


node 1
node 3
node 2
B
A


Frames
Objects
Global frame
Node
node1
node2
node3
Node class
__init__
function
__init__(self, data, next)
default arguments:
next
None
Node instance
data
"A"
next
None
Node instance
data
"B"
next
Node instance
data
"C"
next

¿Qué hicimos?


Frames
Objects
Global frame
Node
head
count
5
Node class
__init__
function
__init__(self, data, next)
default arguments:
next
None
Node instance
data
1
next
None
Node instance
data
2
next
Node instance
data
3
next
Node instance
data
4
next
Node instance
data
5
next

Crear singly linked list

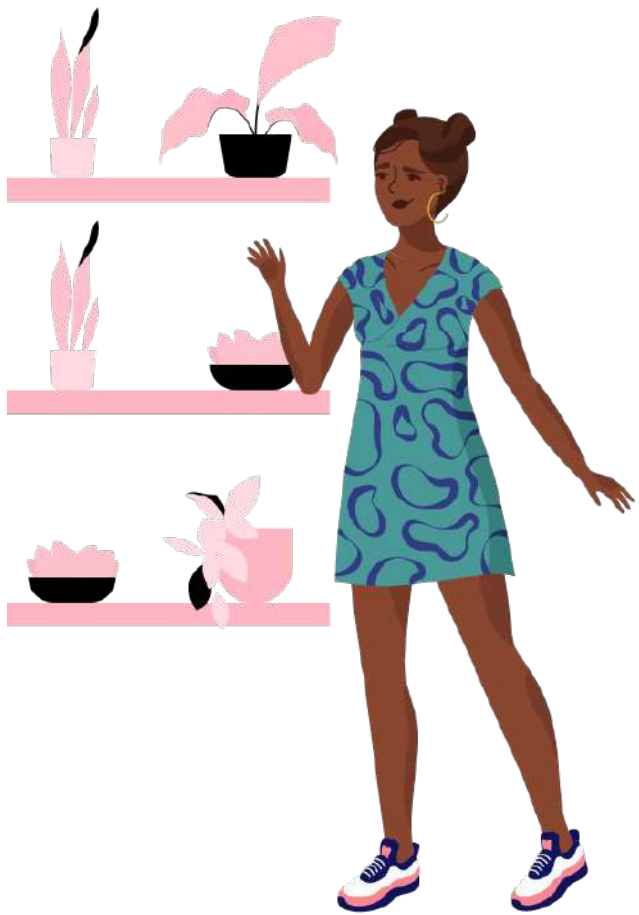
Reto

- Crear un array unidimensional.
- Transfiere los datos a una linked structure sencilla.

Operaciones en single linked structures

El dilema

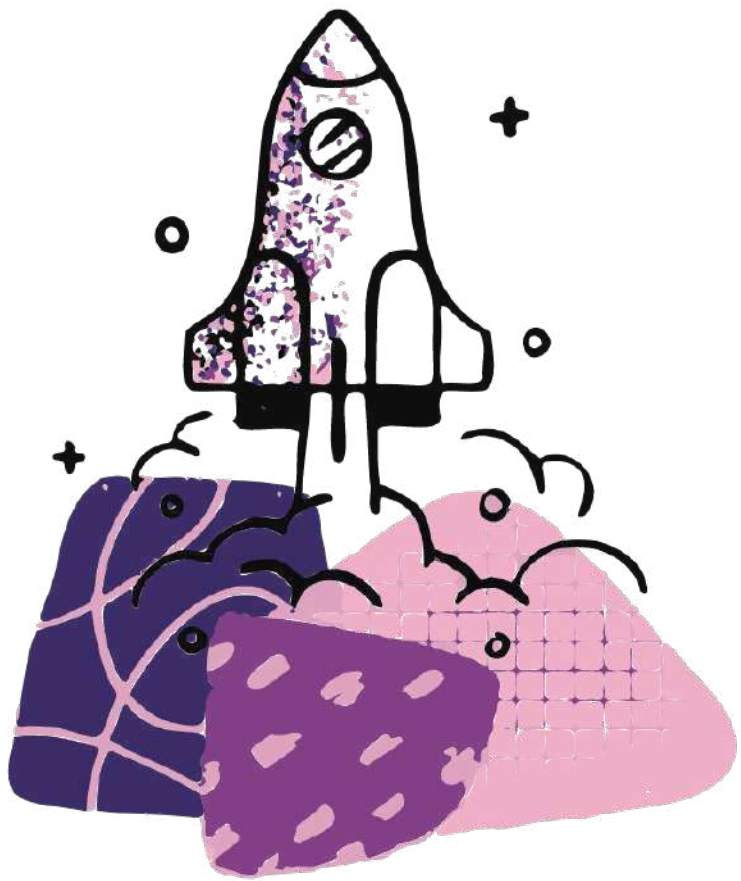
- Los arrays usan índices.
- En linked structures debemos emularlos.



Recorrido

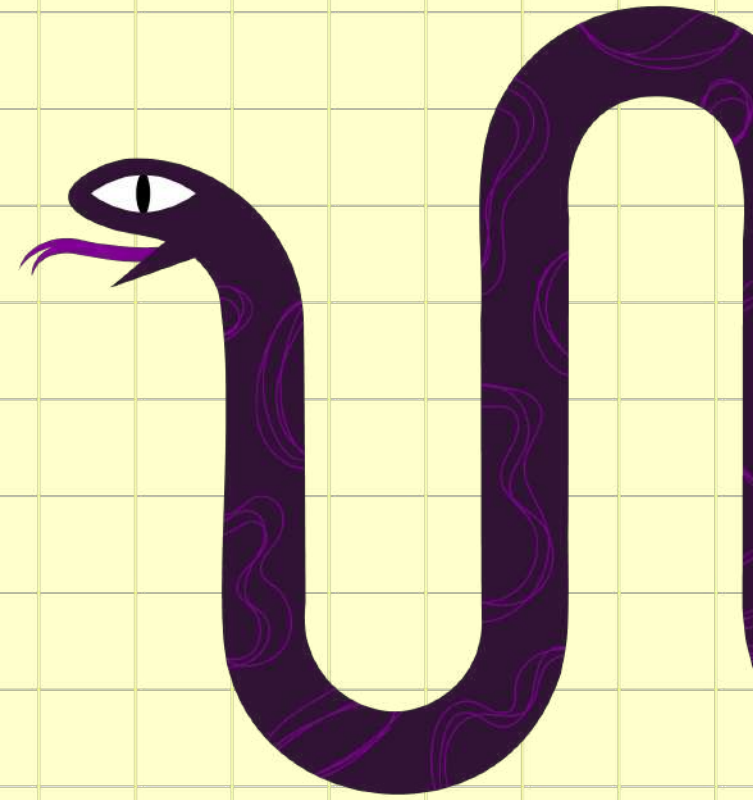
🐍 Variable auxiliar *probe*

- Puntero temporal.
- Recorre la lista consultando datos.

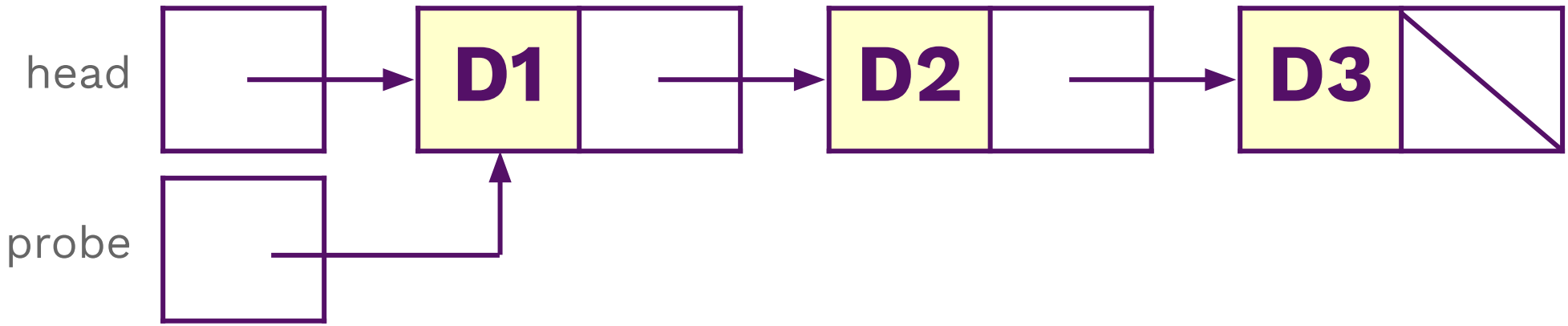


Operaciones a detalle

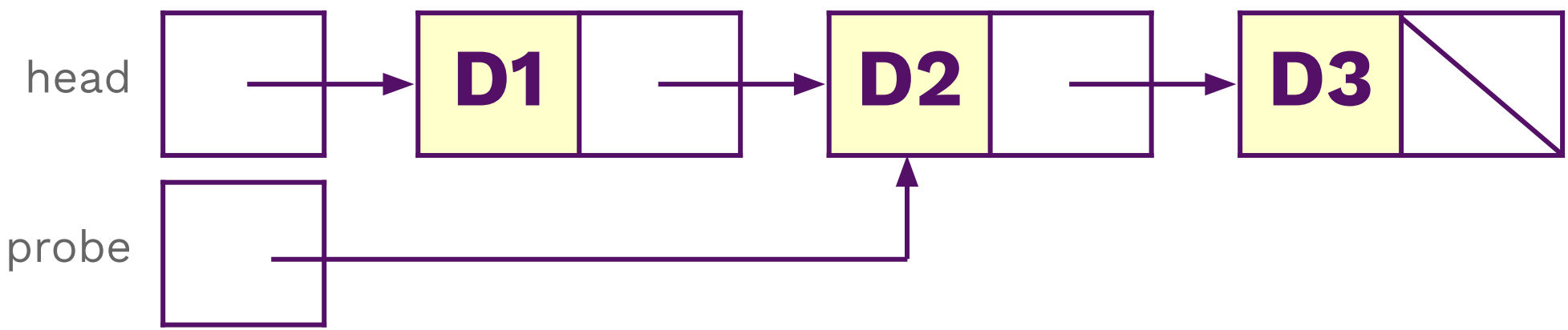
¿Qué sucedió?



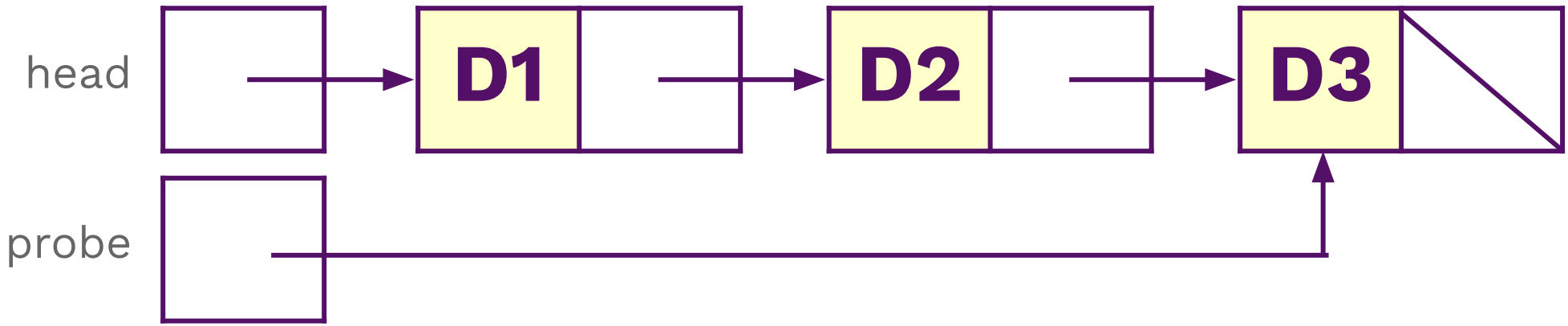
Inicio del paso 1: Visita al nodo D1



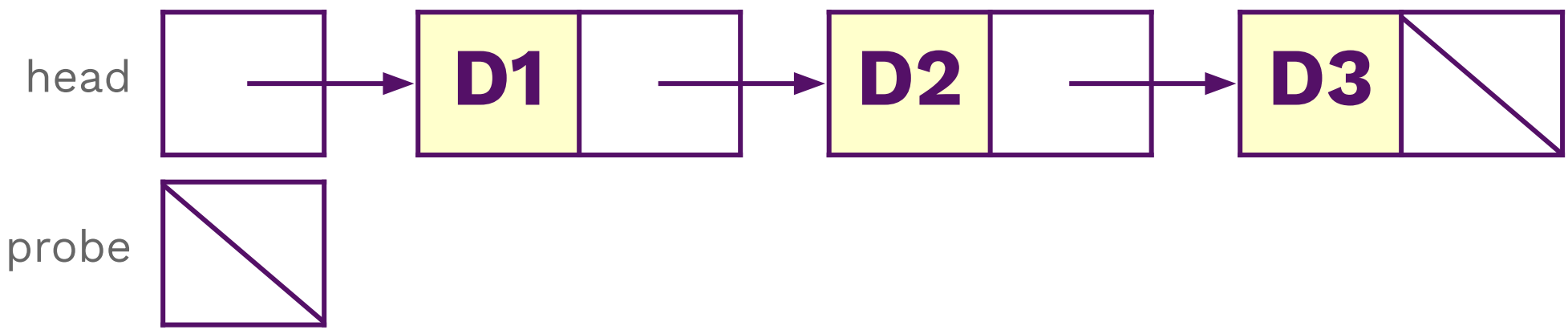
Inicio del paso 2: Visita al nodo D2

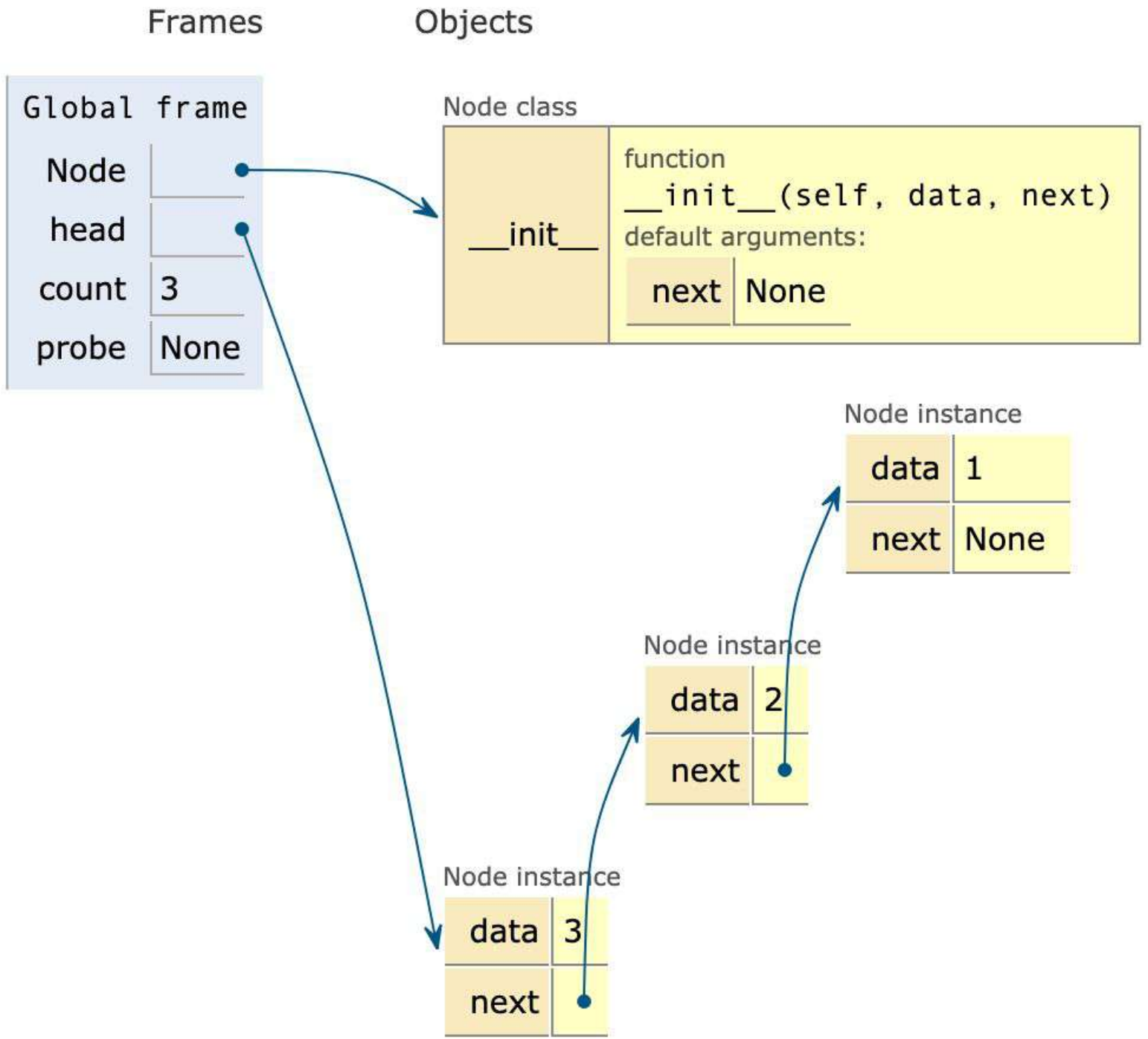


Inicio del paso 3: Visita al nodo D3

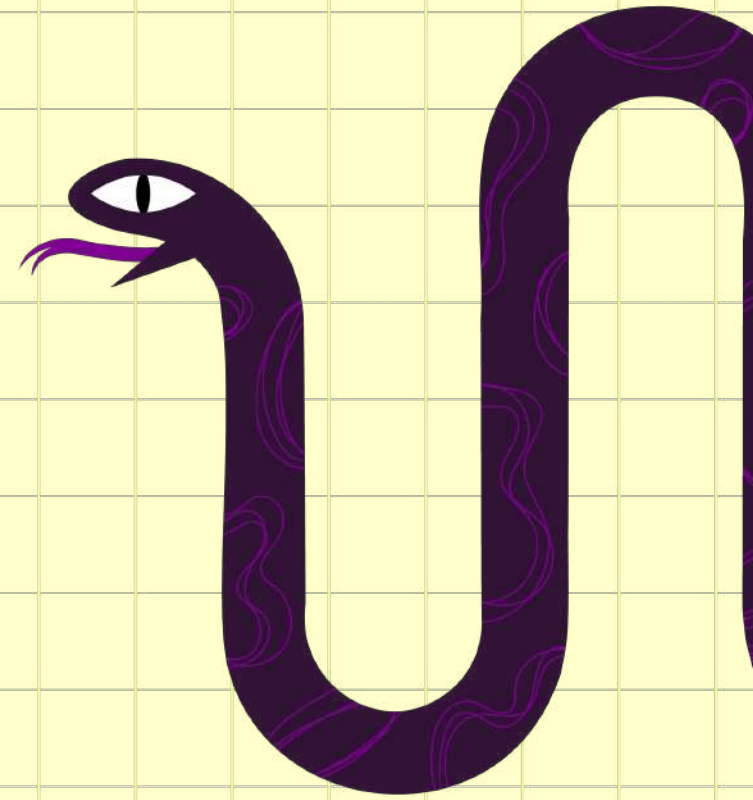


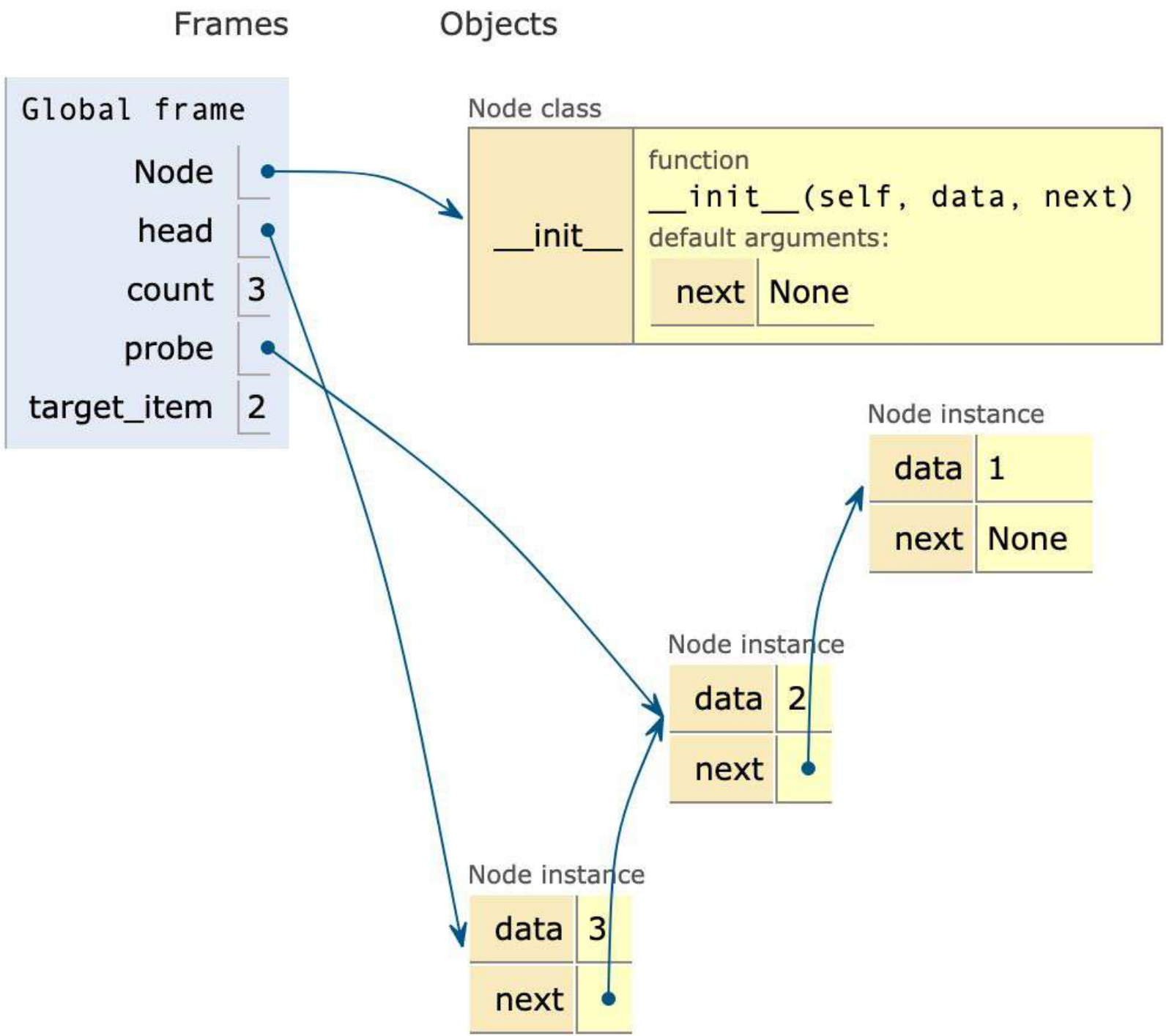
Fin del paso 3: probe es None, el bucle termina



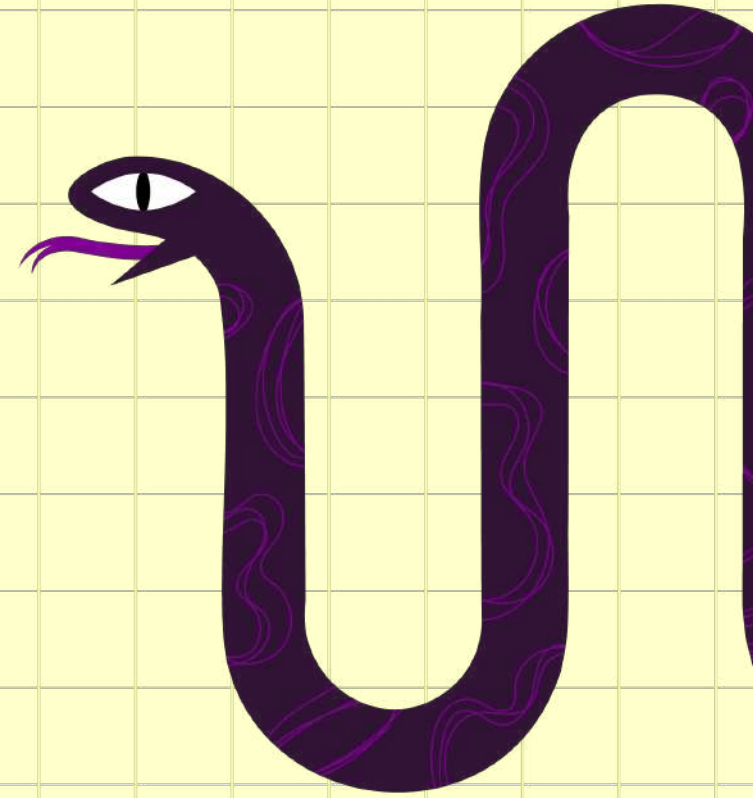

Frames
Objects
Global frame
Node
head
count
3
probe
None
Node class
__init__
function
__init__(self, data, next)
default arguments:
next
None
Node instance
data
1
next
None
Node instance
data
2
next
Node instance
data
3
next

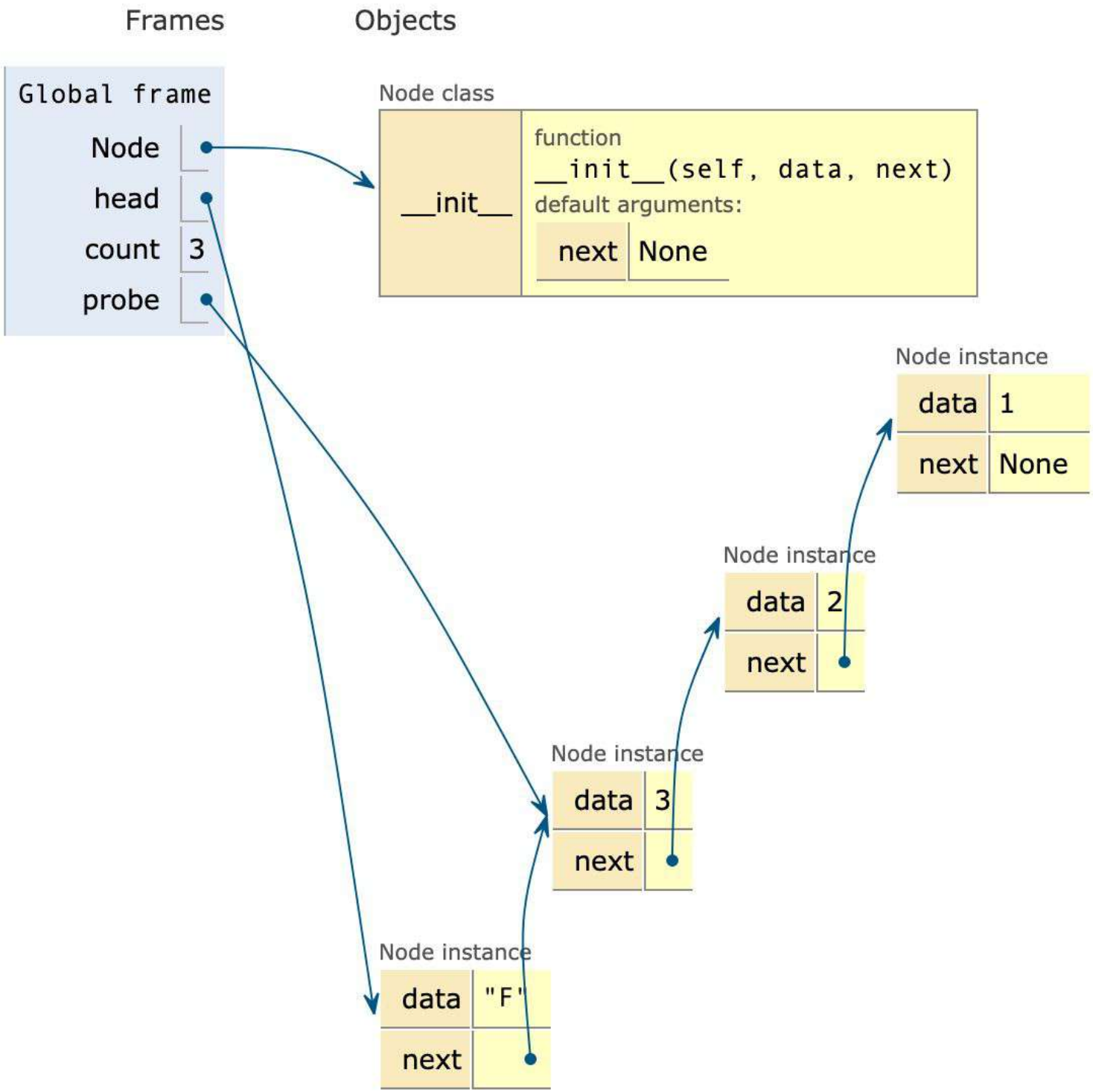
Búsqueda



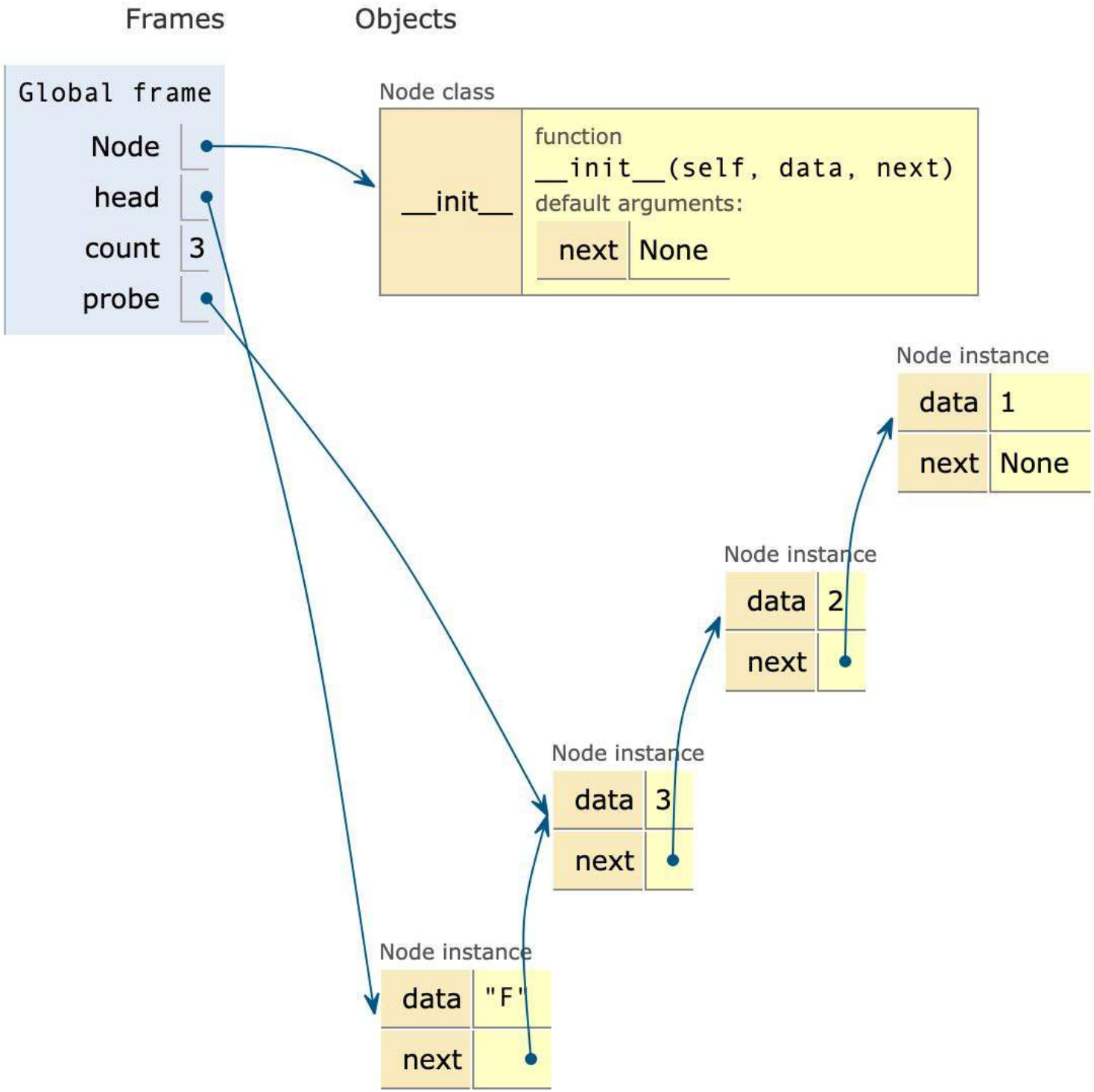

Frames
Objects
Global frame
Node
head
count
3
probe
target_item
2
Node class
__init__
function
__init__(self, data, next)
default arguments:
next
None
Node instance
data
1
next
None
Node instance
data
2
next
Node instance
data
3
next

Reemplazo

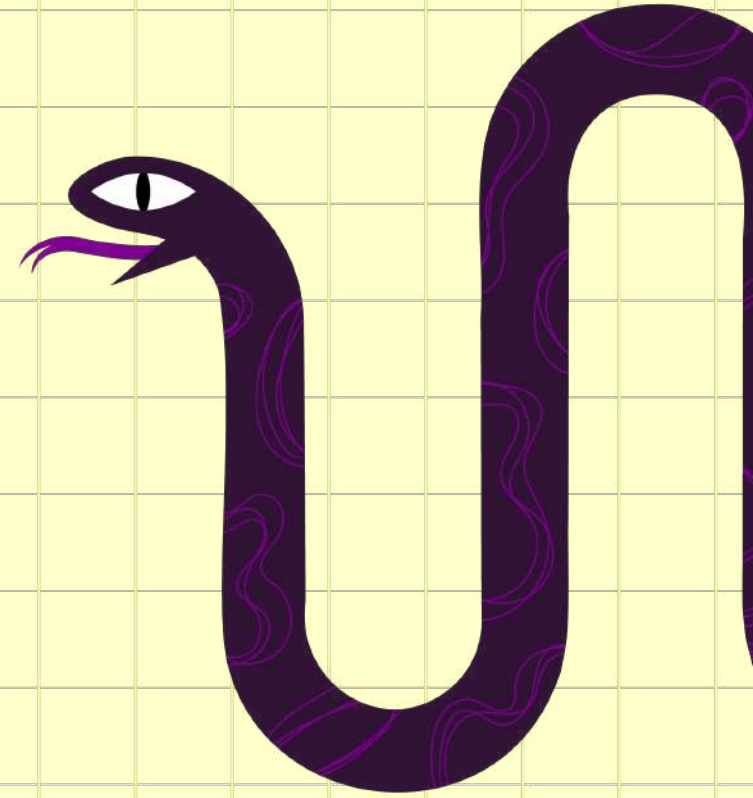


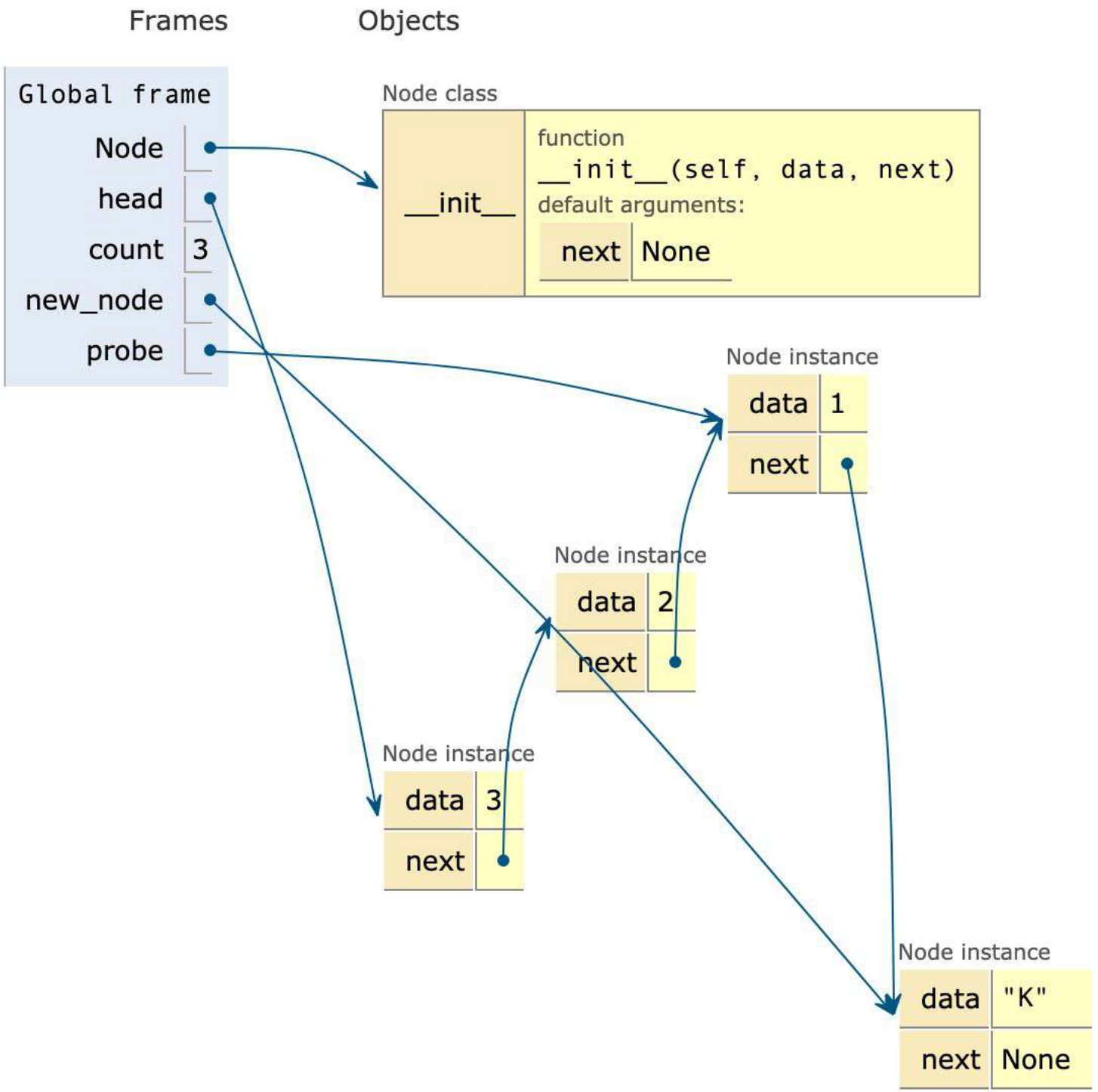

Frames
Objects
Global frame
Node
head
count
3
probe
Node class
__init__
function
__init__(self, data, next)
default arguments:
next
None
Node instance
data
1
next
None
Node instance
data
2
next
Node instance
data
3
next
Node instance
data
"F"
next

Insertar al inicio

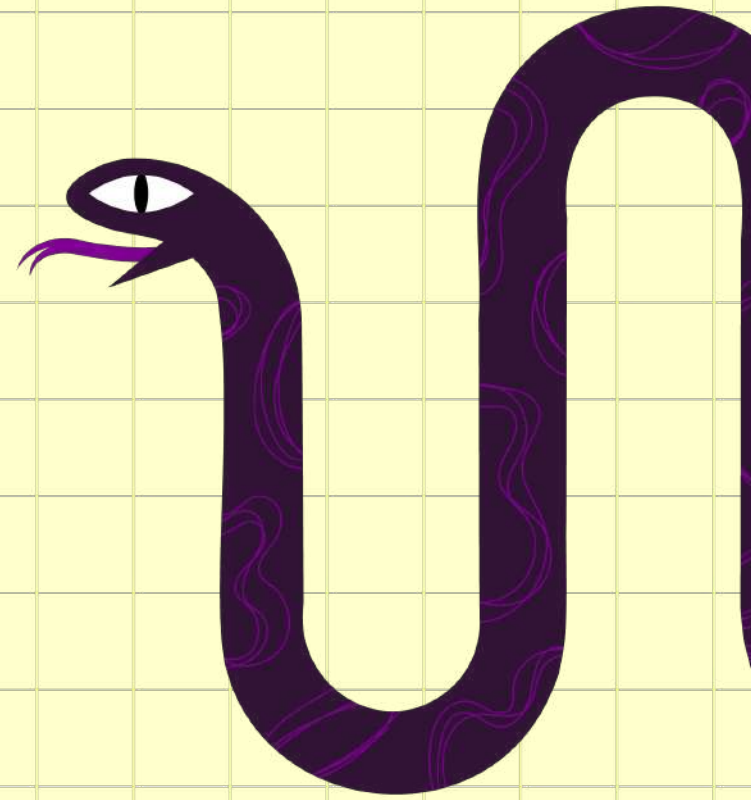

Frames
Objects
Global frame
Node
head
count
3
probe
Node class
__init__
function
__init__(self, data, next)
default arguments:
next
None
Node instance
data
1
next
None
Node instance
data
2
next
Node instance
data
3
next
Node instance
data
"F"
next

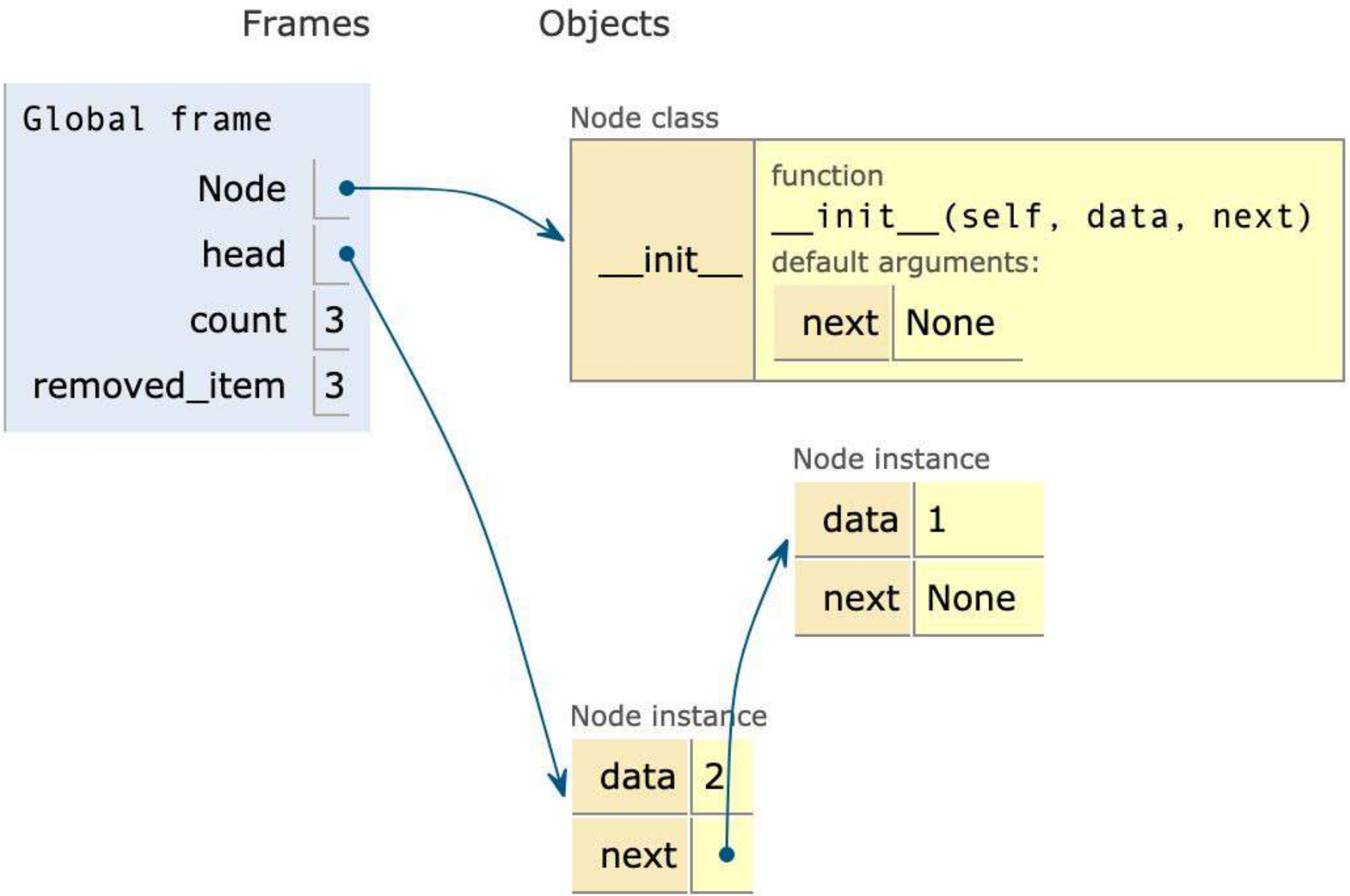
Insertar al final



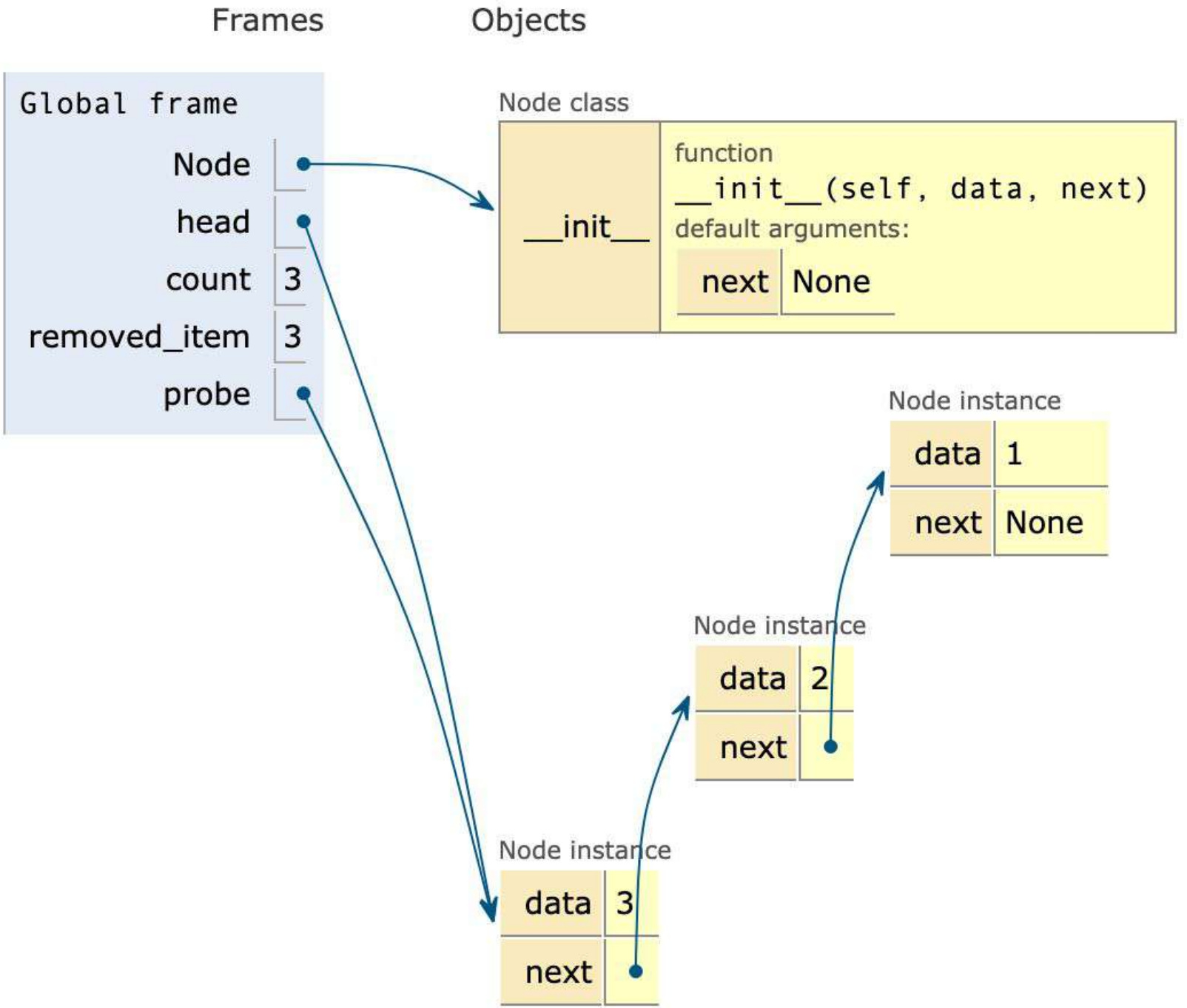

Frames
Objects
Global frame
Node
head
count
3
new_node
probe
Node class
__init__
function
__init__(self, data, next)
default arguments:
next
None
Node instance
data
1
next
Node instance
data
2
next
Node instance
data
3
next
Node instance
data
"K"
next
None

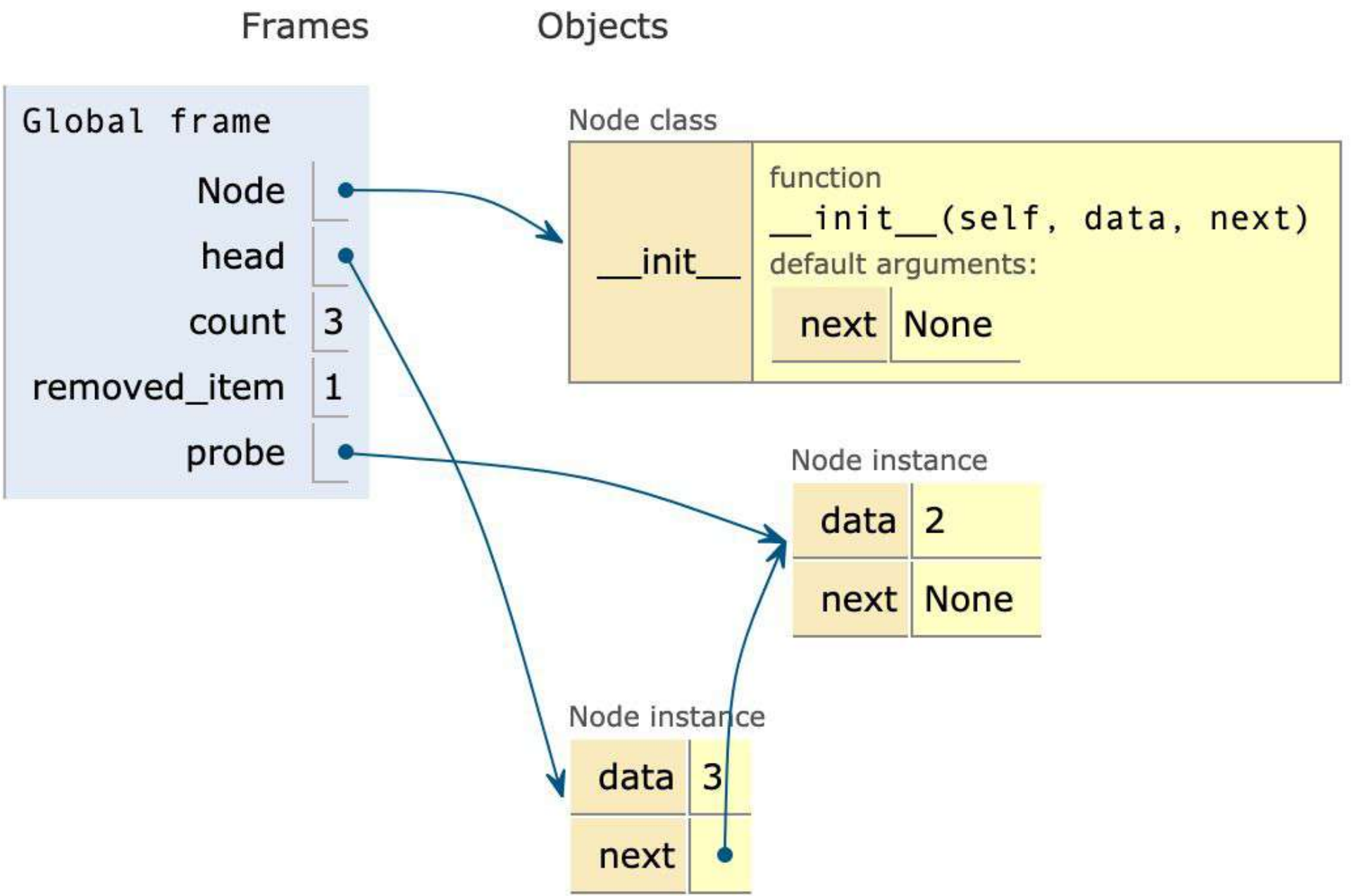
Eliminar primer nodo



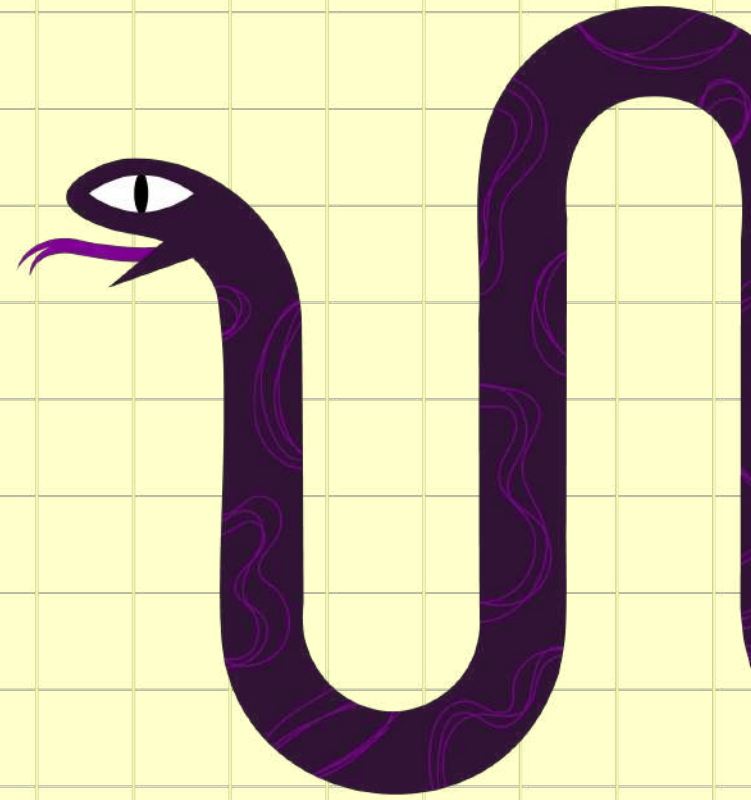

Frames
Objects
Global frame
Node
head
count
3
removed_item
3
Node class
__init__
function
__init__(self, data, next)
default arguments:
next
None
Node instance
data
1
next
None
Node instance
data
2
next

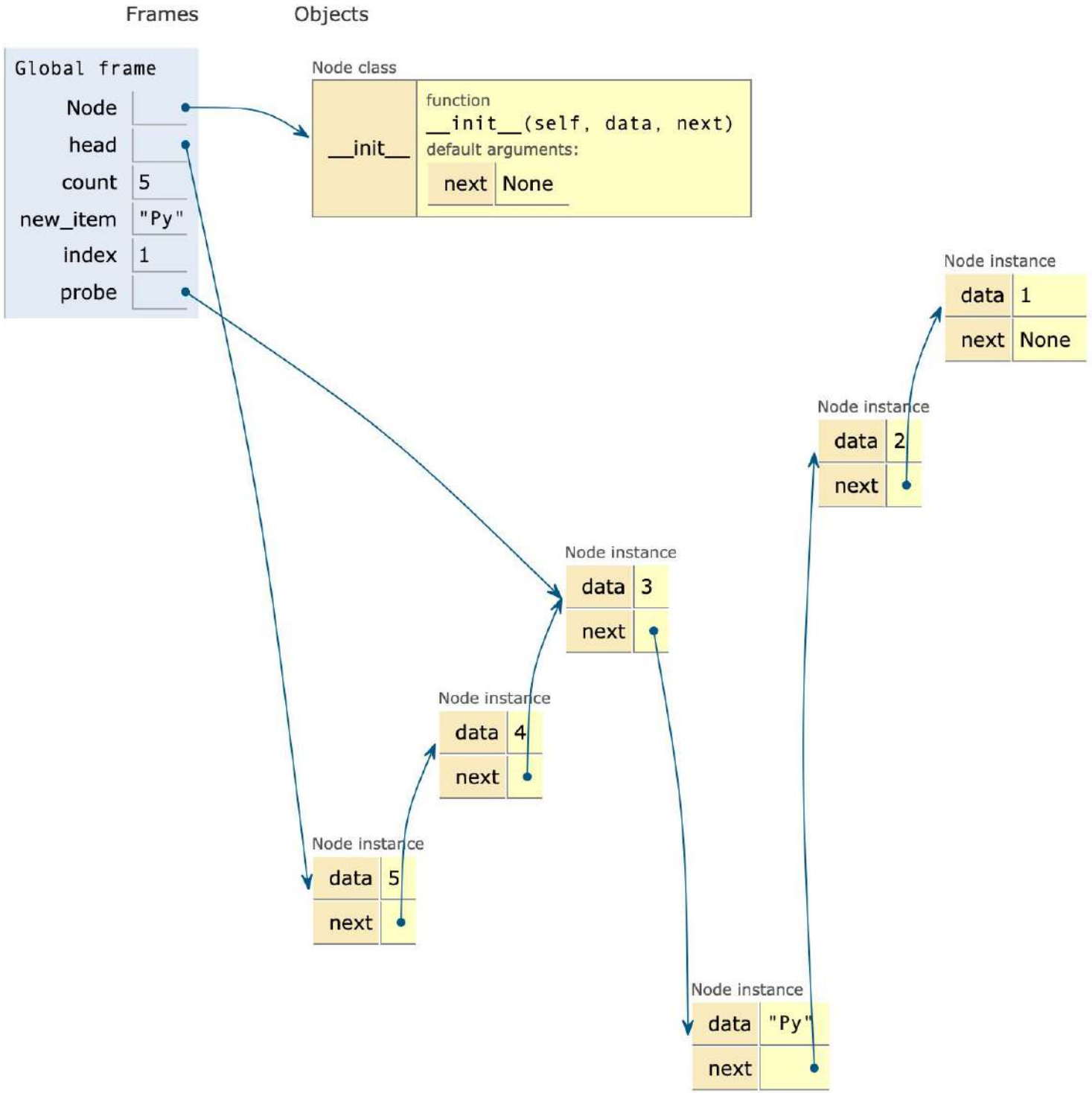
Eliminar último nodo


Frames
Objects
Global frame
Node
head
count
3
removed_item
3
probe
Node class
__init__
function
__init__(self, data, next)
default arguments:
next
None
Node instance
data
1
next
None
Node instance
data
2
next
Node instance
data
3
next

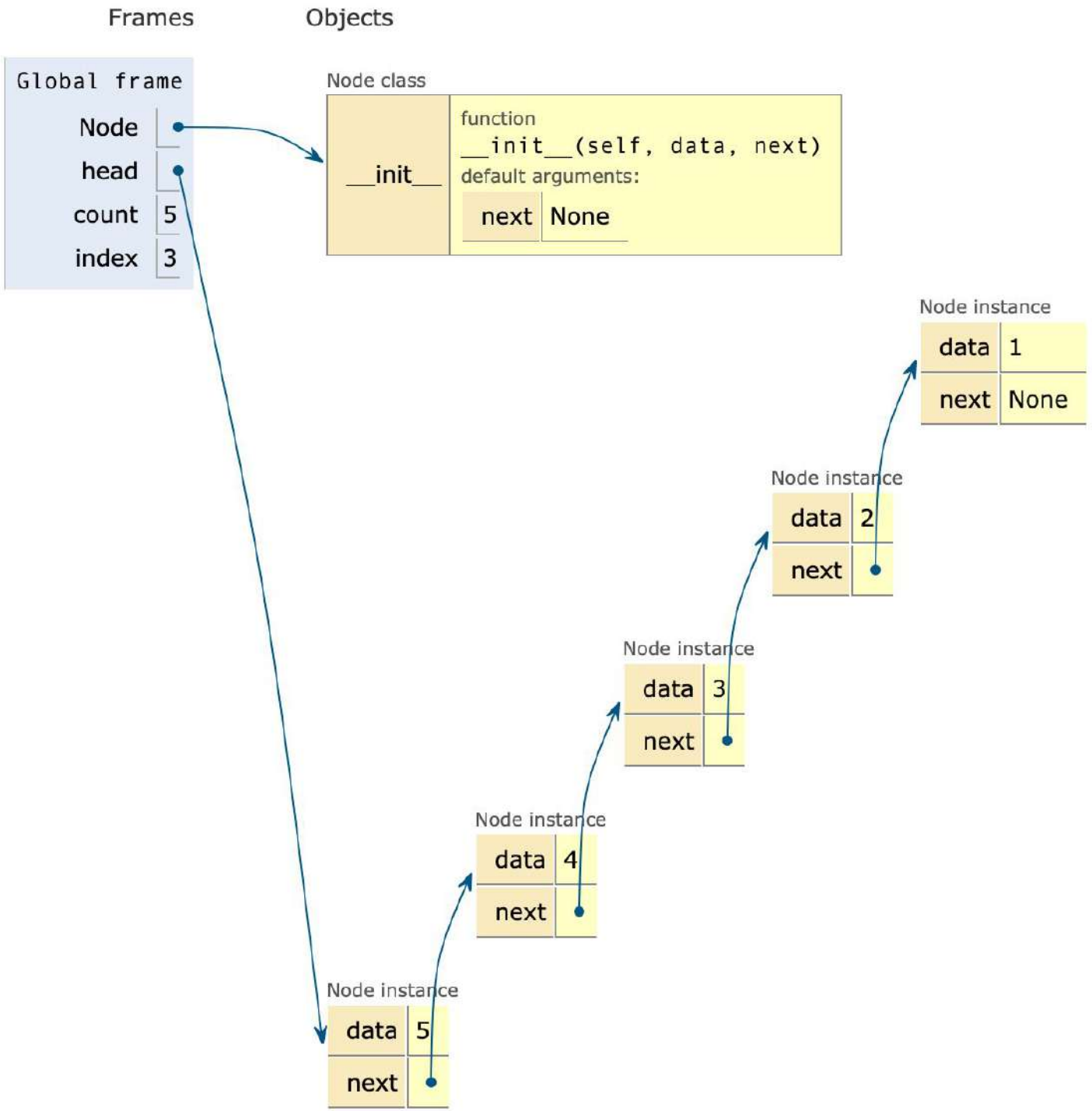

Frames
Objects
Global frame
Node
head
count
3
removed_item
1
probe
Node class
__init__
function
__init__(self, data, next)
default arguments:
next
None
Node instance
data
2
next
None
Node instance
data
3
next

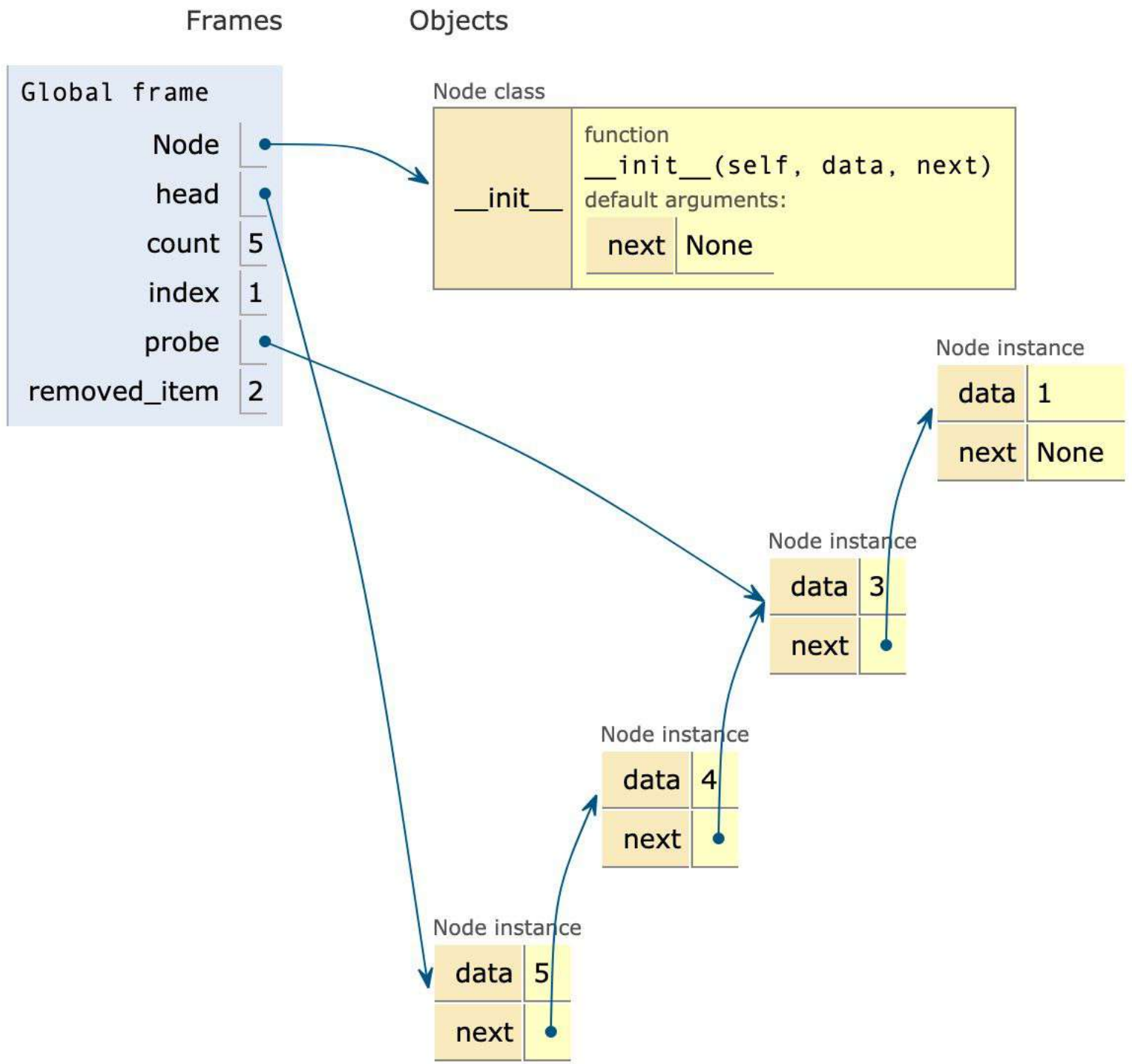
Insertar en cualquier posición




Frames
Objects
Global frame
Node
head
count
5
new_item
"Py"
index
1
probe
Node class
__init__
function
__init__(self, data, next)
default arguments:
next
None
Node instance
data
1
next
None
Node instance
data
2
next
Node instance
data
3
next
Node instance
data
4
next
Node instance
data
5
next
Node instance
data
"Py"
next

Eliminar en cualquier posición

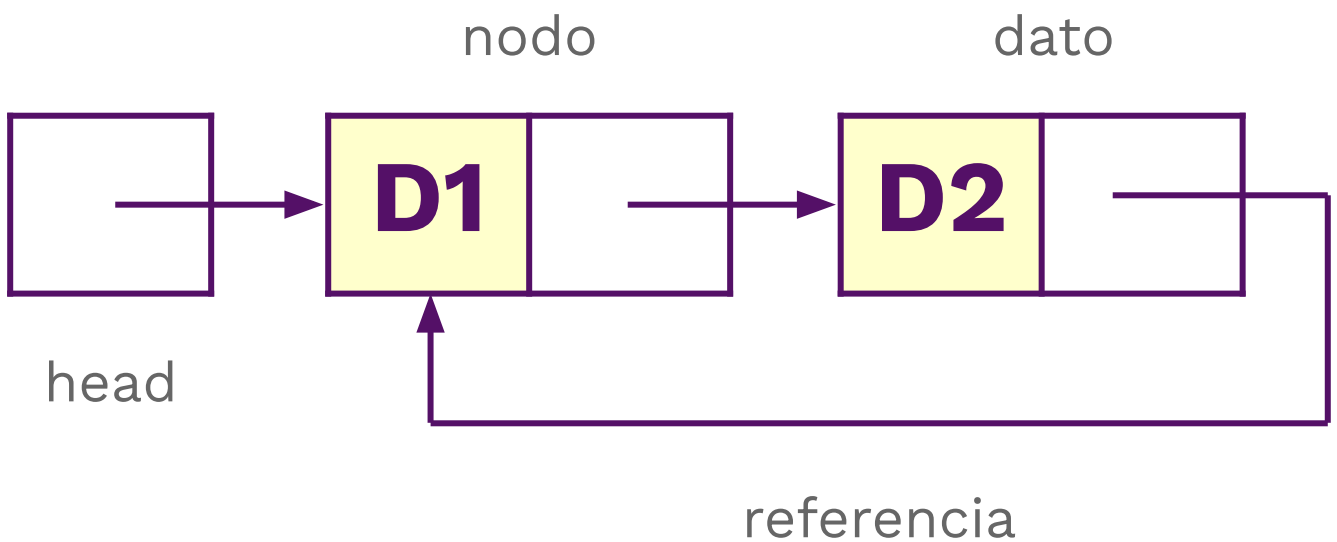

Frames
Objects
Global frame
Node
head
count 5
index 3
Node class
__init__
function
__init__(self, data, next)
default arguments:
next None
Node instance
data 1
next None
Node instance
data 2
next
Node instance
data 3
next
Node instance
data 4
next
Node instance
data 5
next

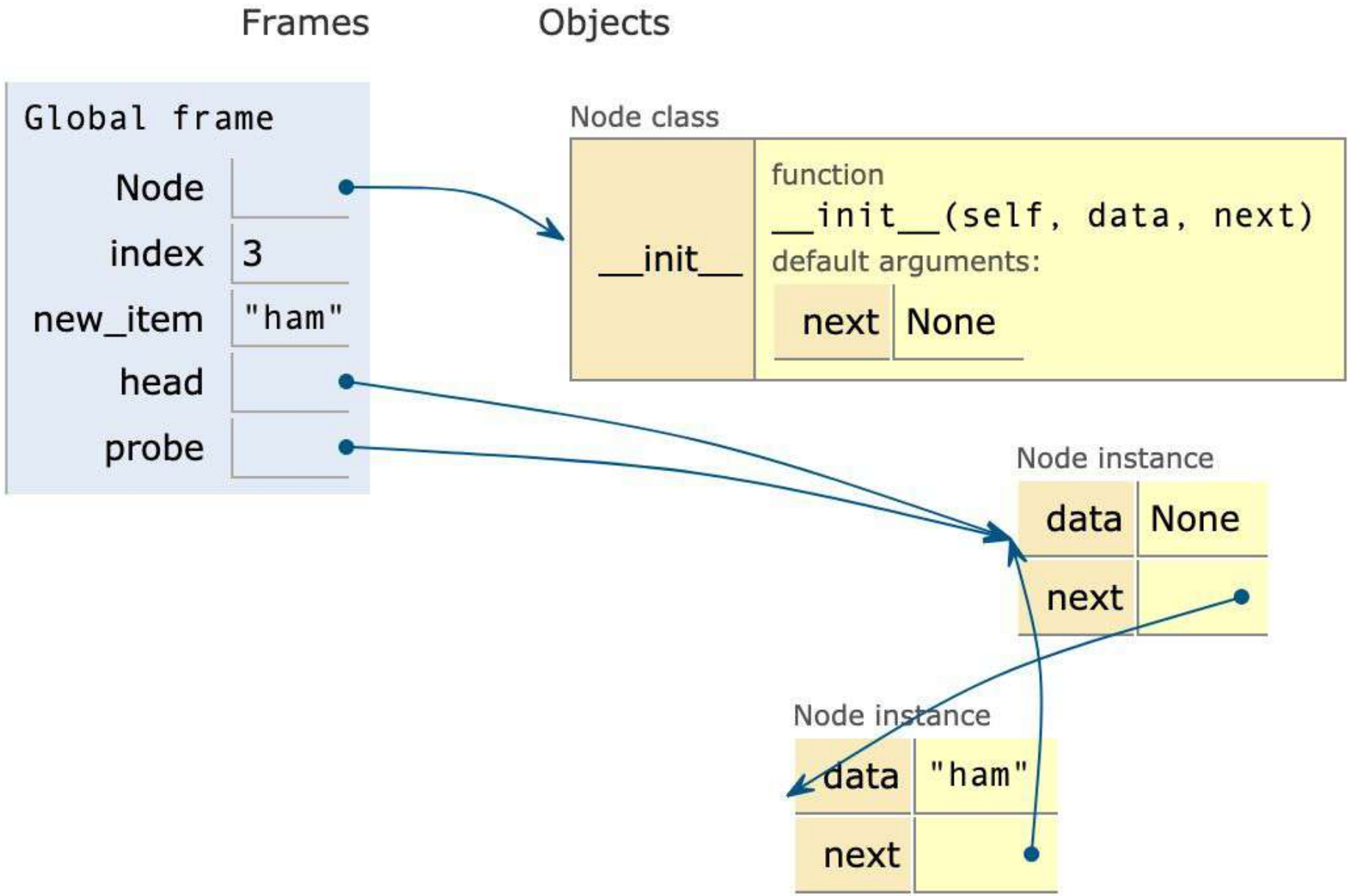

Frames
Objects
Global frame
Node
head
count
5
index
1
probe
removed_item
2
Node class
__init__
function
__init__(self, data, next)
default arguments:
next
None
Node instance
data
1
next
None
Node instance
data
3
next
Node instance
data
4
next
Node instance
data
5
next

Circular Linked List

Singly Linked List

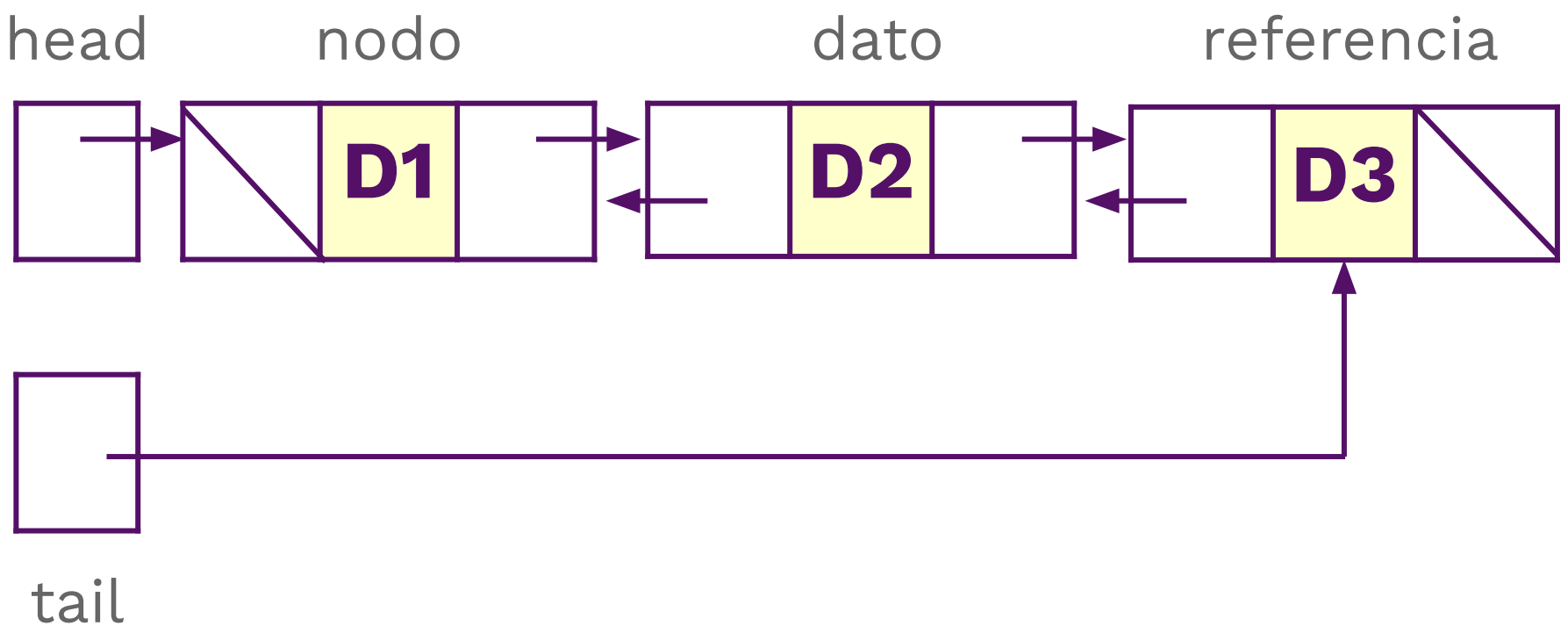
Circular Linked Structure

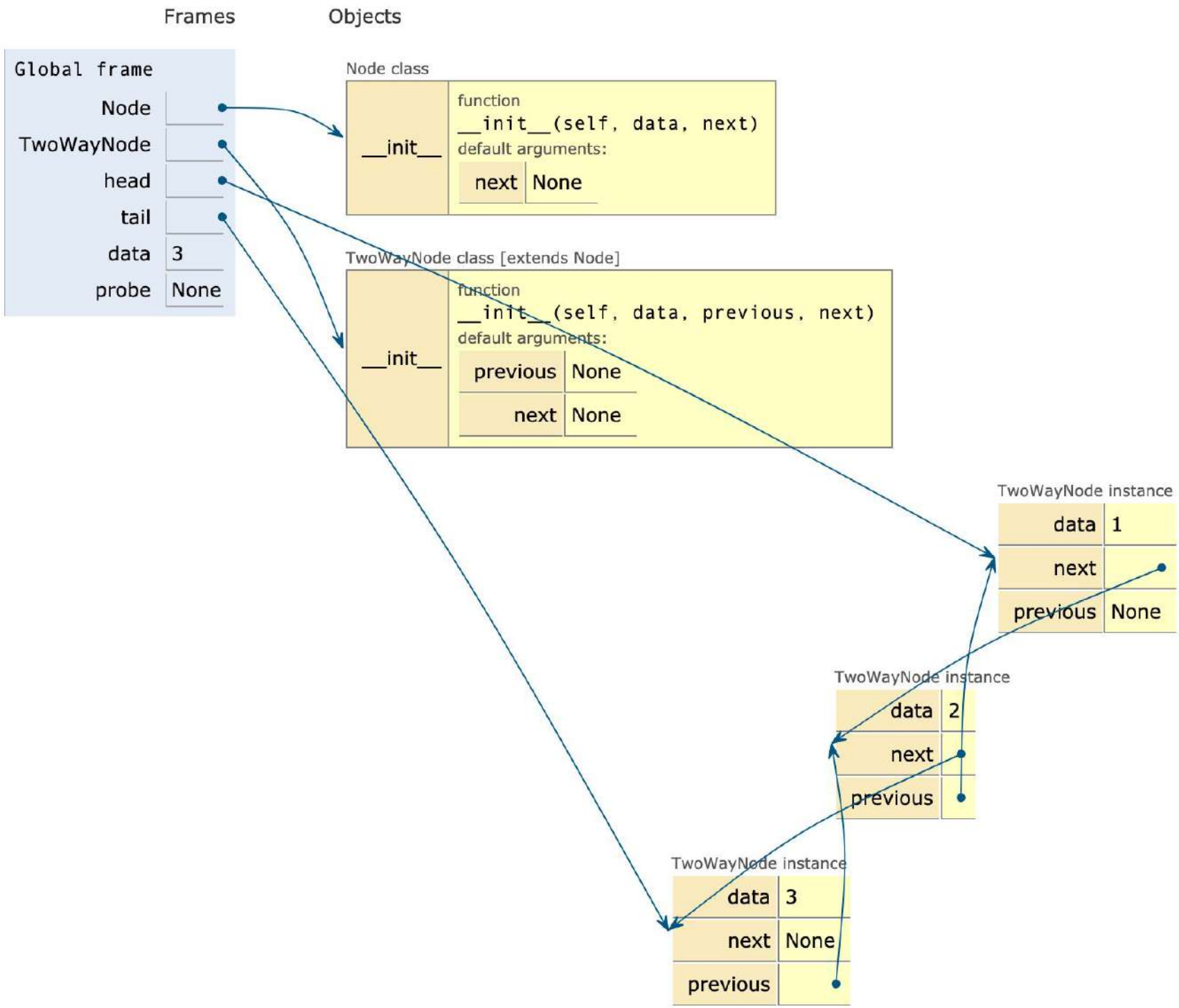



Frames
Objects
Global frame
Node
index
3
new_item
"ham"
head
probe
Node class
__init__
function
__init__(self, data, next)
default arguments:
next
None
Node instance
data
None
next
Node instance
data
"ham"
next

Double Linked List

Double Linked List




Frames
Objects
Global frame
Node
TwoWayNode
head
tail
data 3
probe None
Node class
__init__
function
__init__(self, data, next)
default arguments:
next None
TwoWayNode class [extends Node]
__init__
function
__init__(self, data, previous, next)
default arguments:
previous None
next None
TwoWayNode instance
data 1
next
previous None
TwoWayNode instance
data 2
next
previous
TwoWayNode instance
data 3
next None
previous

Reto

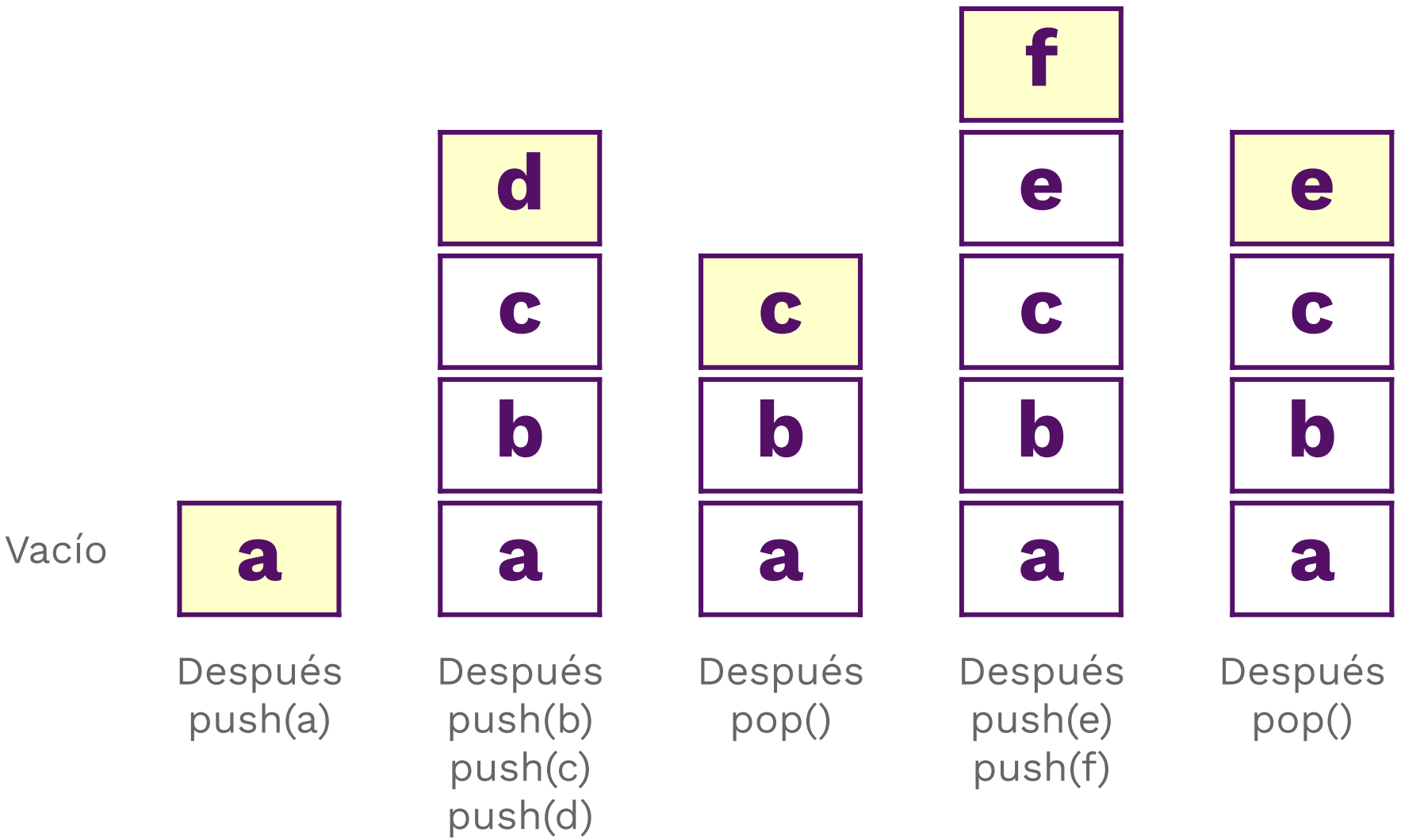
- Crea un doubly circular linked list con 5 nodos.
- Agrega métodos de operación ya vistos.

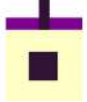
¿Qué son stacks?

¿Qué son?

- Colección lineal
- Basados en arrays o linked lists
- LIFO (last-in first-out)
- Añadir: push
- Remover: pop
- Top
- Bottom

Stack


Vacío
a
Después push(a)
d
c
b
a
Después push(b) push(c) push(d)
c
b
a
Después pop()
f
e
c
b
a
Después push(e) push(f)
e
c
b
a
Después pop()



Aplicaciones

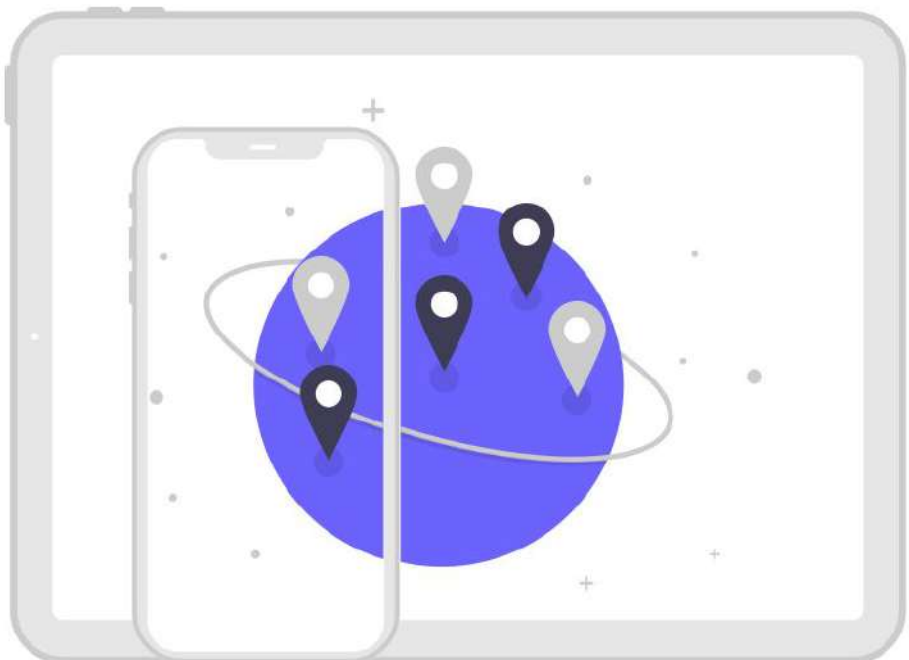
Convertir operaciones *infix* a *postfix*

4 x (5 - (7 + 2)) -> 4 5 7 2 + - *

-16

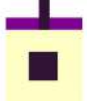
Aplicaciones

- Invertir el orden de una lista.
- Implementar “undo”.
- Mantener historiales.
- Backtracking.



Stack vs. list

- Similares, no iguales.
- List: append, pop.
- Es afectada por sus otros métodos.

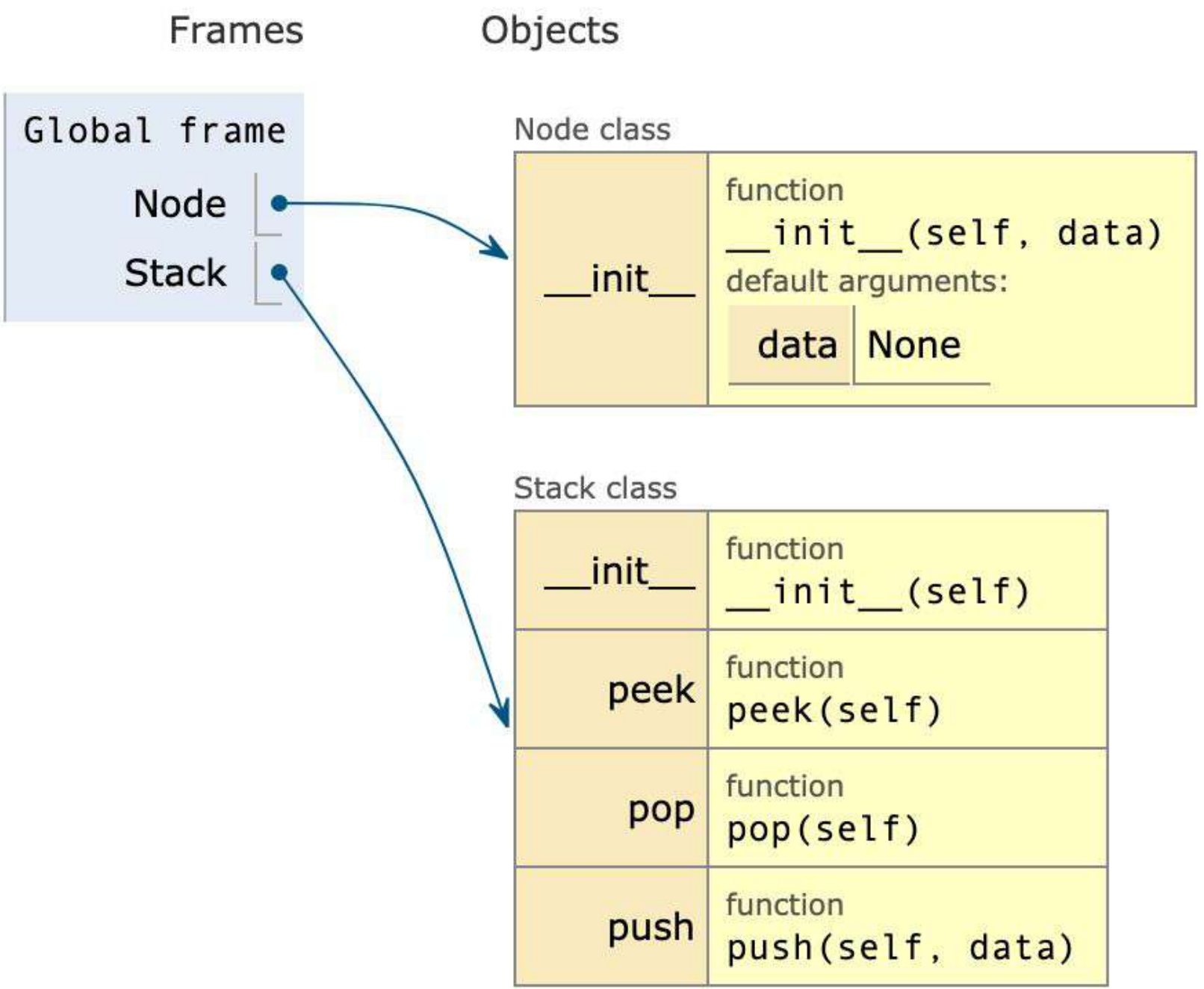


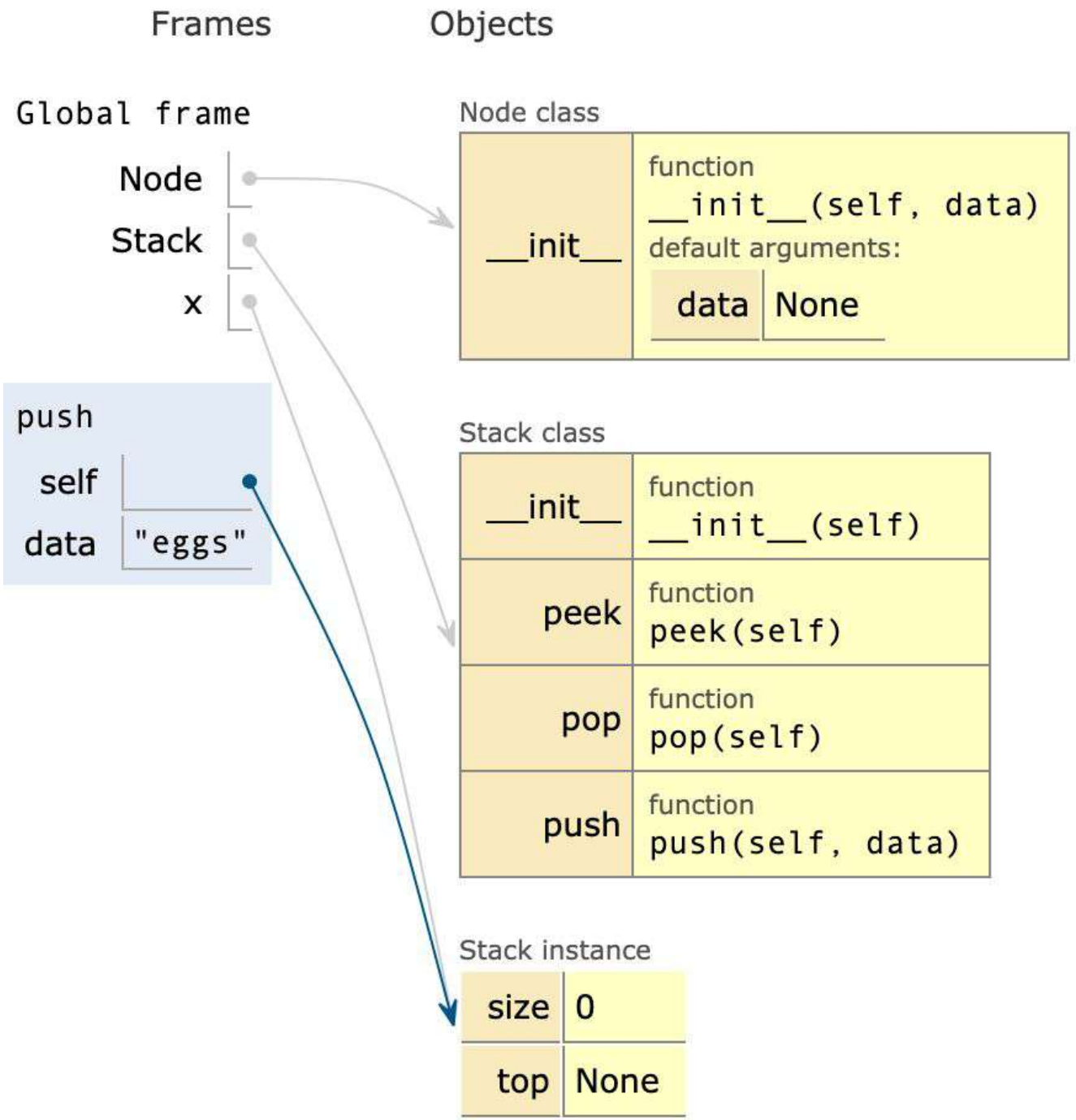
Métodos de stacks

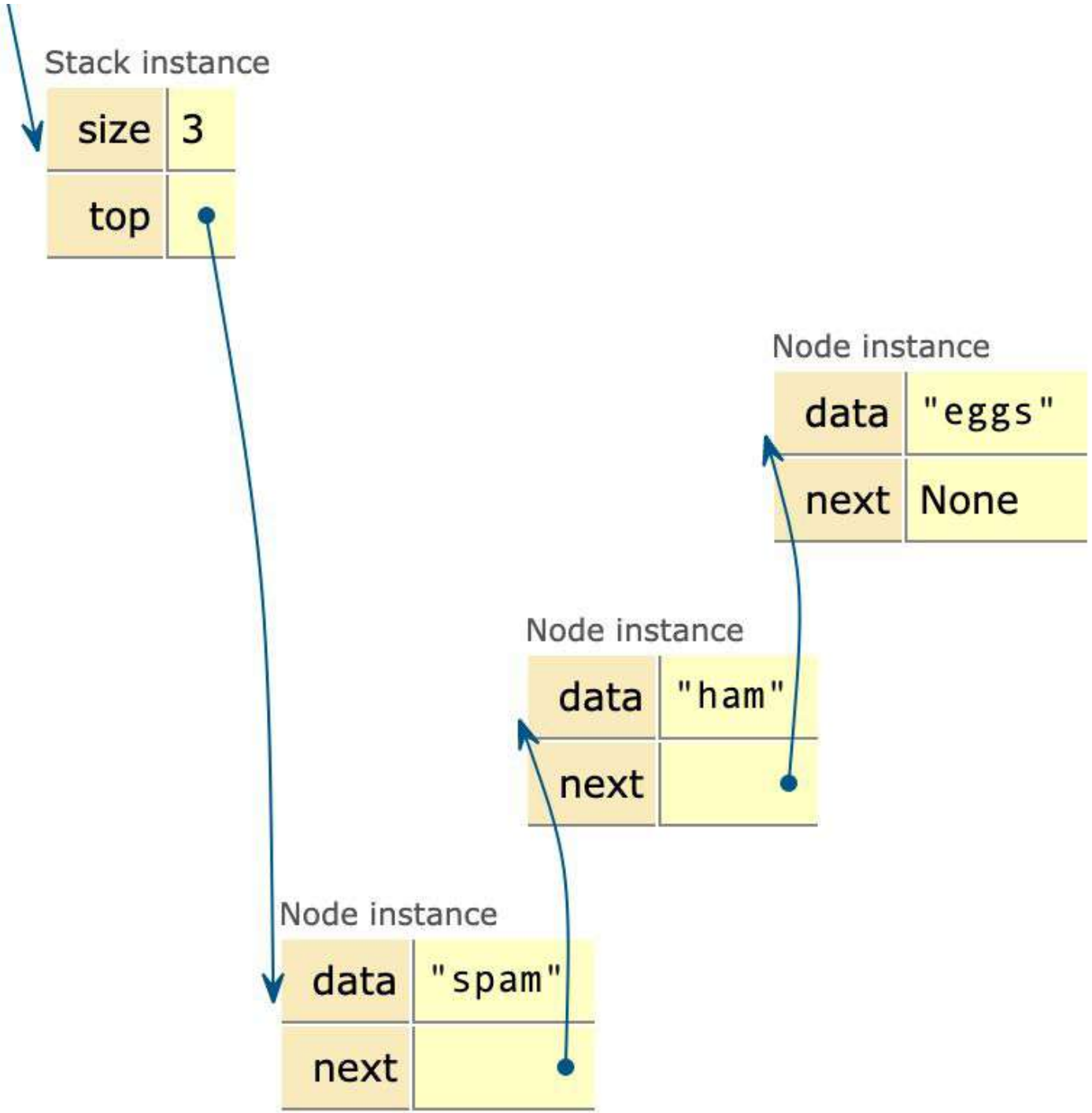
Método	Qué hace
stack.is_empty()	Retorna True si está vacío, *False* en caso contrario.
stack.__len__()	Retorna número de elementos.
stack.__str__()	Retorna la representación *string* de un elemento.
stack.__iter__()	Visita todos los elementos de *top* a *bottom*.
stack.__contains__(item)	Retorna True si hay un elemento, *False* en caso contrario.
stack.__add__(stack2)	Retorna un nuevo stack con los elementos de stack y stack2.
stack.clear()	Vacía el stack.
stack.peek()	Retorna el elemento *top* del stack.
stack.push(item)	Agregar un elemento en el *top* del stack.
stack.pop()	Remueve y retorna el elemento en *top* del stack.

Crear un stack

¿Qué hicimos?


Frames
Objects
Global frame
Node
Stack
Node class
__init__
function
__init__(self, data)
default arguments:
data
None
Stack class
__init__
function
__init__(self)
peek
function
peek(self)
pop
function
pop(self)
push
function
push(self, data)


Frames
Objects
Global frame
Node
Stack
x
push
self
data
"eggs"
Node class
__init__
function
__init__(self, data)
default arguments:
data
None
Stack class
__init__
function
__init__(self)
peek
function
peek(self)
pop
function
pop(self)
push
function
push(self, data)
Stack instance
size
0
top
None


Stack instance
size
3
top
Node instance
data
"eggs"
next
None
Node instance
data
"ham"
next
Node instance
data
"spam"
next

Reto #1

- Crea una clase stack.
- Añade los métodos para:
- Retornar/buscar un elemento.
- Recorrer el stack.
- Vaciar el stack.

Reto #2

- Crea una clase Stack que herede de una clase Array.
- Crea los métodos de Stack.

Queues


OPEN
Fred's
AWARD WINNING
OPEN
Fred's
55
Est. 1983
The Gift Box
RESTAURANT
Award Winning
Award Winning

Conceptos importantes

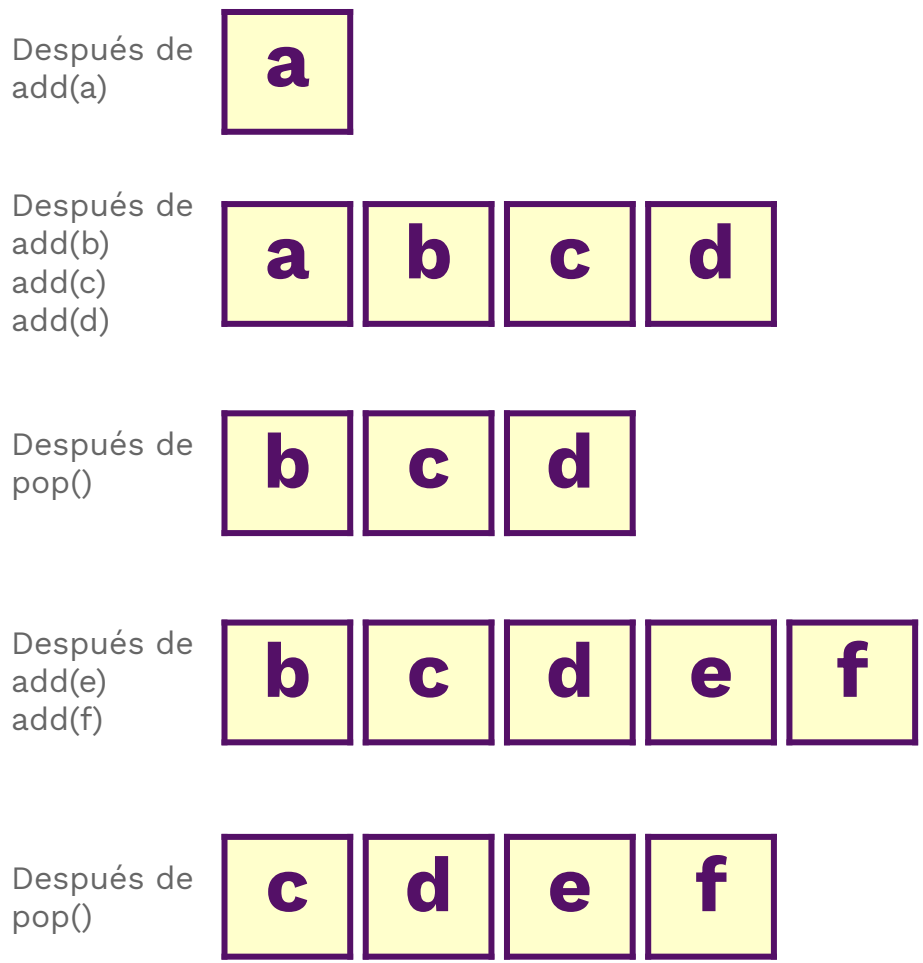
- **FIFO:** First-In, First-Out.
- **Rear:** último elemento.
- **Front:** primer elemento.

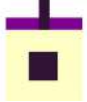
Conceptos importantes

- **Priority queues:** se basa en FIFO con elementos de mayor/menor prioridad.

Operaciones fundamentales

- **pop()**: remover front.
- **add():** añadir a rear.



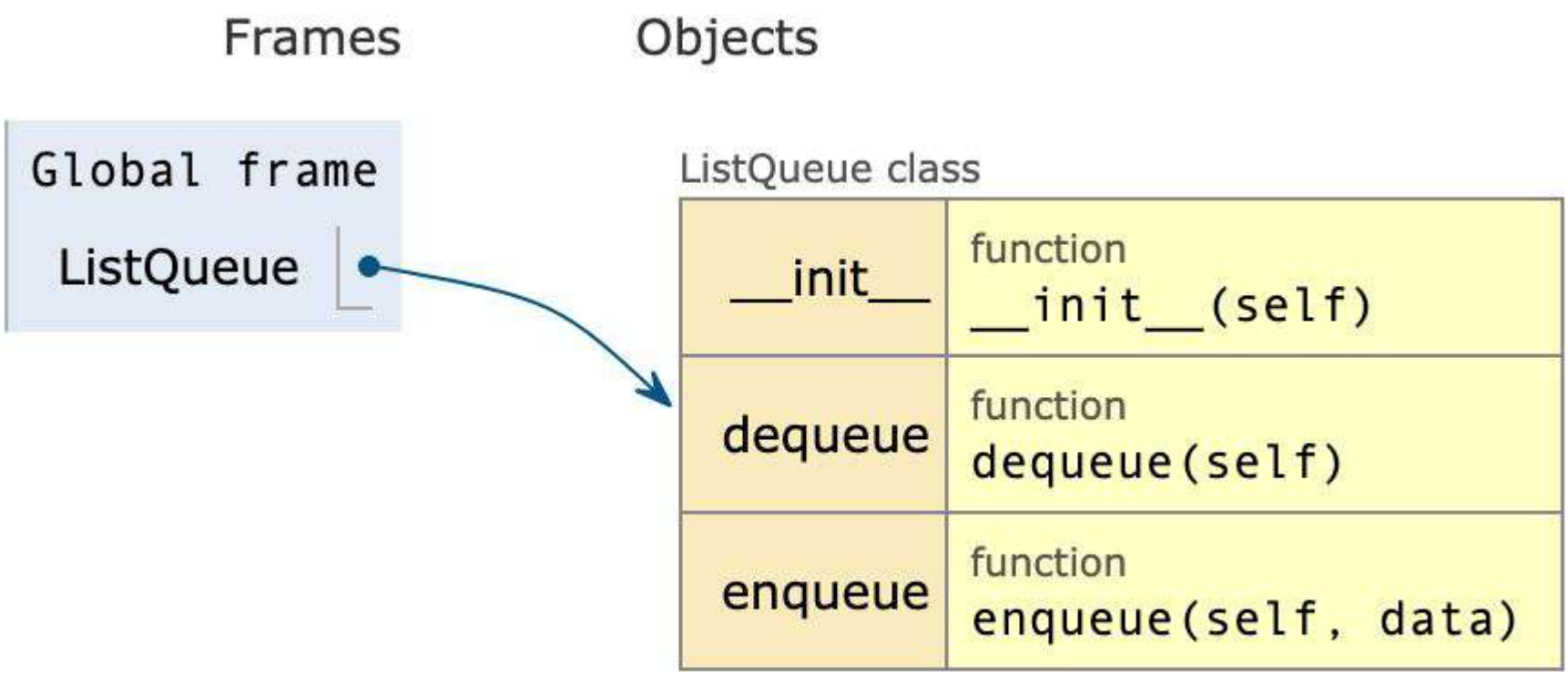


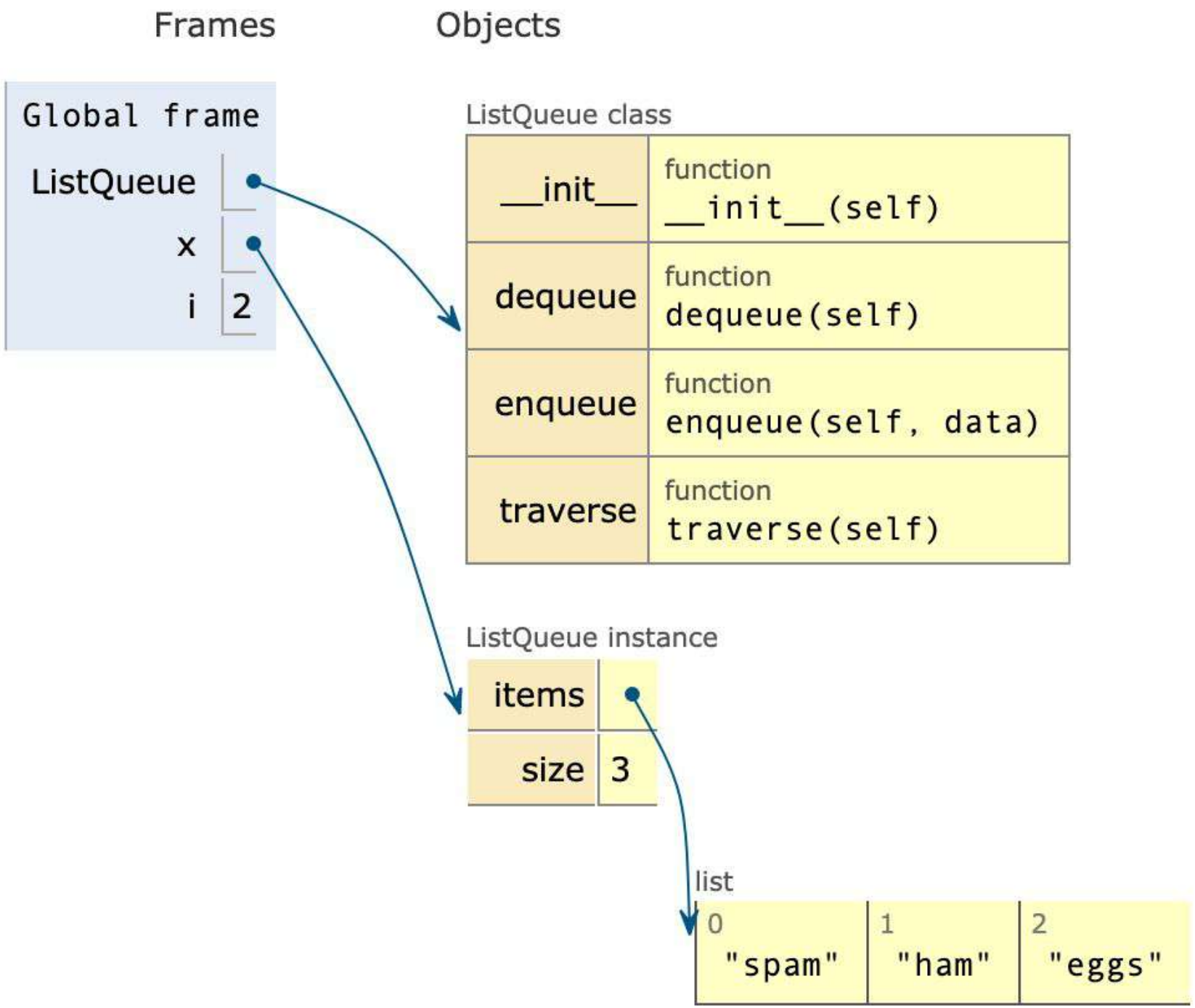
Métodos de queues

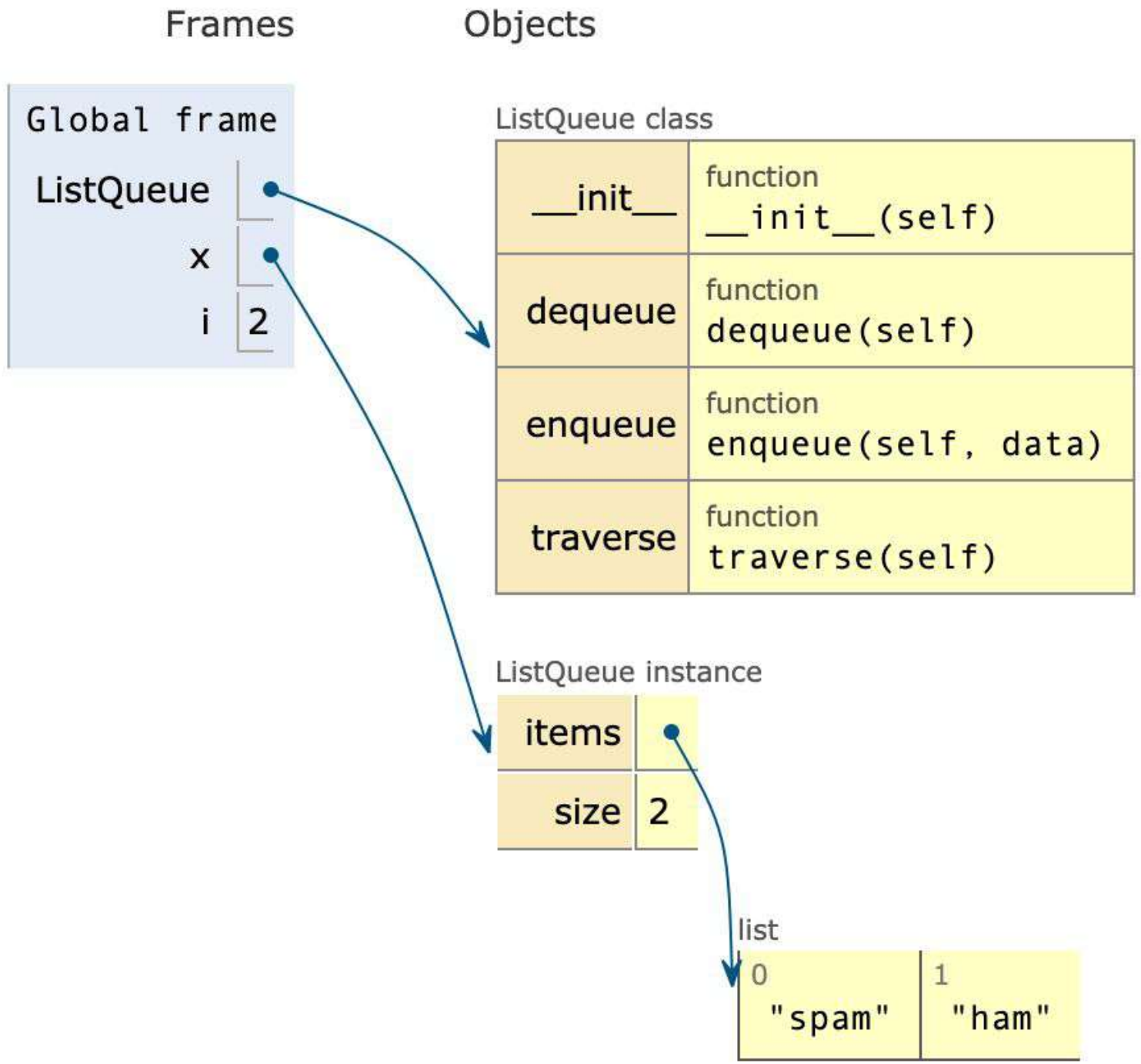
Método	Qué hace
queue.is_empty()	Retorna *True* si está vacía, *False* en caso contrario.
queue.__len__()	Retorna el número de elementos.
queue.__str__()	Retorna la representación *string* de un elemento.
queue.__iter__()	Visita todos los elementos de *front* a *rear.*
queue.__contains__(item)	Retorna True si hay un elemento, *False* en caso contrario.
queue.__add__(queue2)	Retorna un nuevo queue con los elementos de queue y queue2.
queue.clear()	Vacía el queue.
queue.peek()	Retorna el elemento *front* del queue.
queue.add(item)	Agregar un elemento en el *front* del stack.
queue.pop()	Remueve y retorna el elemento en *front* del stack.

Queues basadas en listas

¿Qué hicimos?


Frames
Objects
Global frame
ListQueue
ListQueue class
__init__
function
__init__(self)
dequeue
function
dequeue(self)
enqueue
function
enqueue(self, data)


Frames
Objects
Global frame
ListQueue
x
i
2
ListQueue class
__init__
function
__init__(self)
dequeue
function
dequeue(self)
enqueue
function
enqueue(self, data)
traverse
function
traverse(self)
ListQueue instance
items
size
3
list
0
"spam"
1
"ham"
2
"eggs"

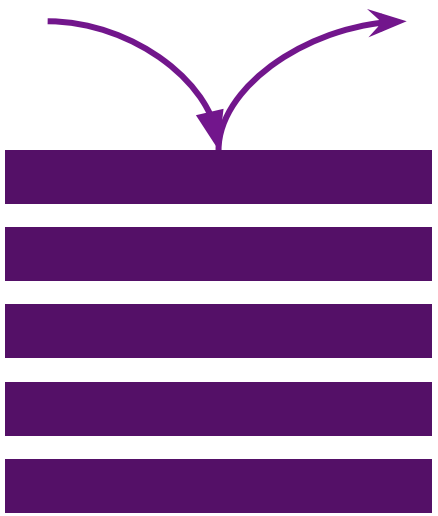

Frames
Objects
Global frame
ListQueue
x
i
2
ListQueue class
__init__
function
__init__(self)
dequeue
function
dequeue(self)
enqueue
function
enqueue(self, data)
traverse
function
traverse(self)
ListQueue instance
items
size
2
list
0
"spam"
1
"ham"

Queue basada en dos stacks

Stack vs. Queue

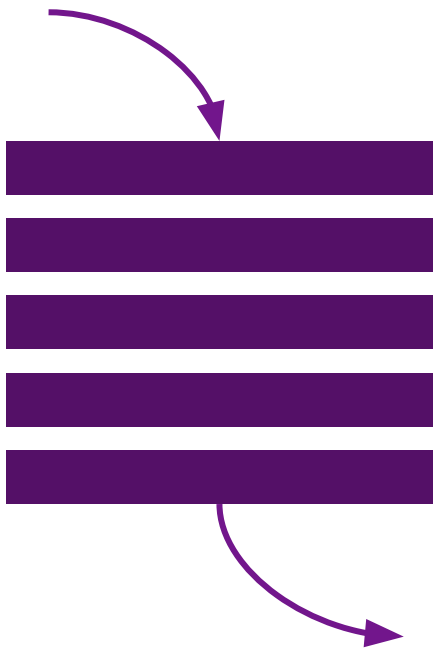
Stack

Last in, first out

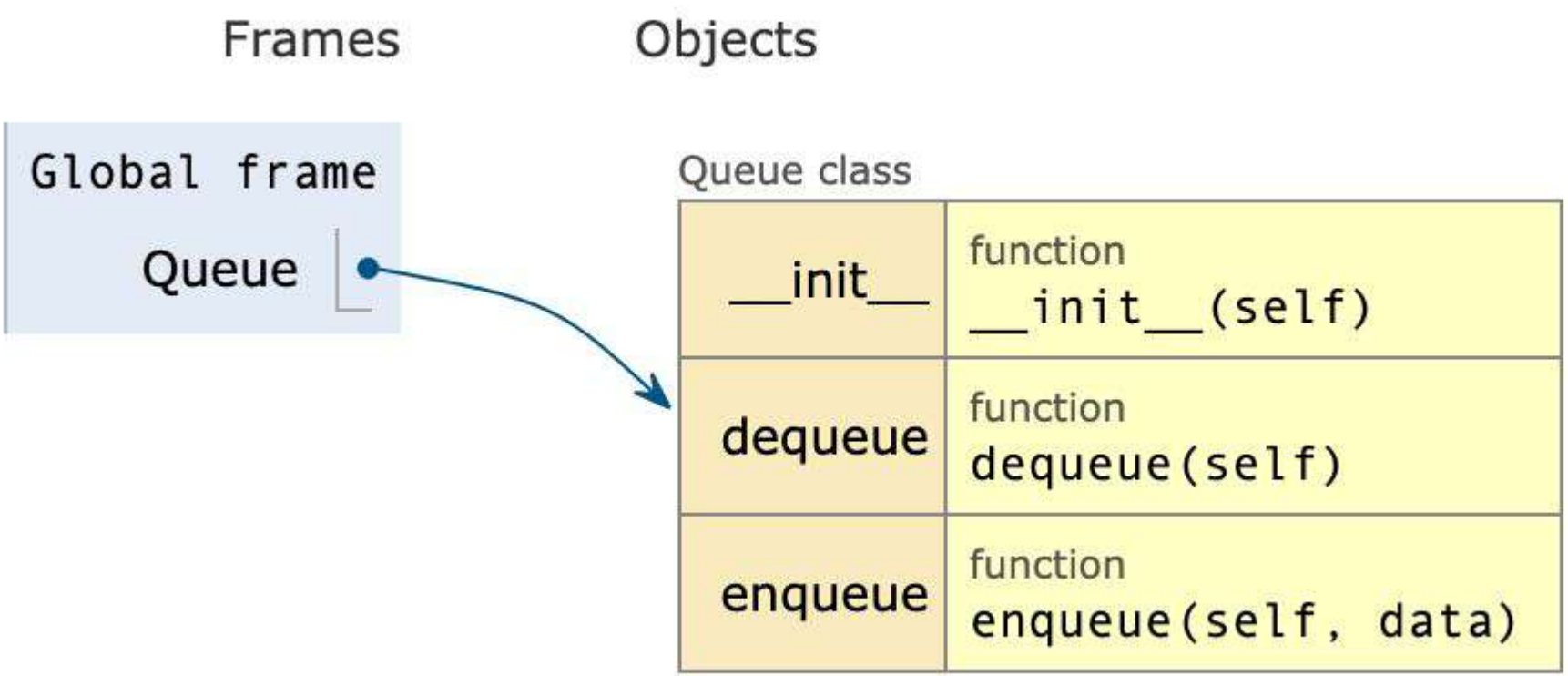


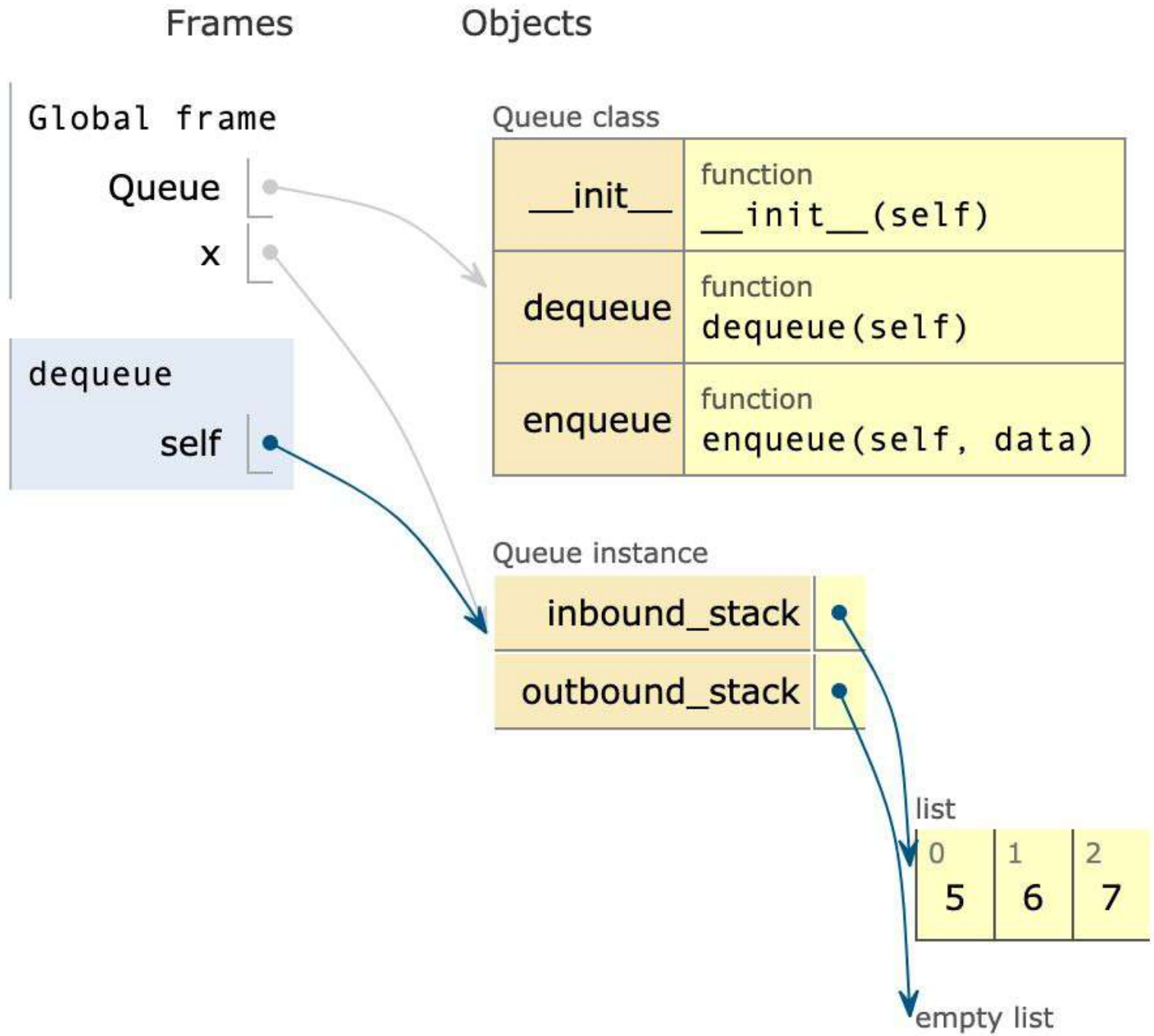
Queue

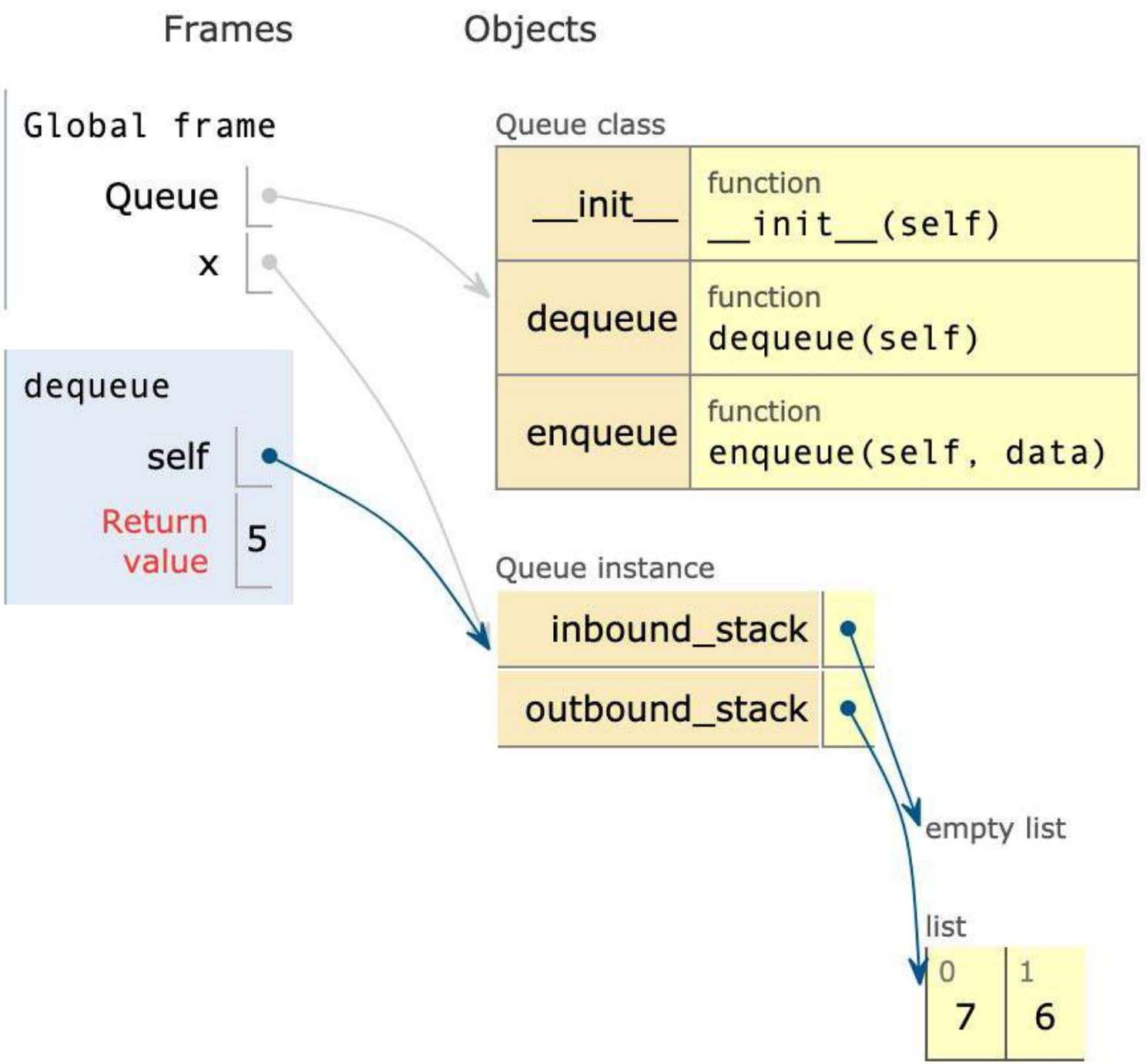
First in, first out

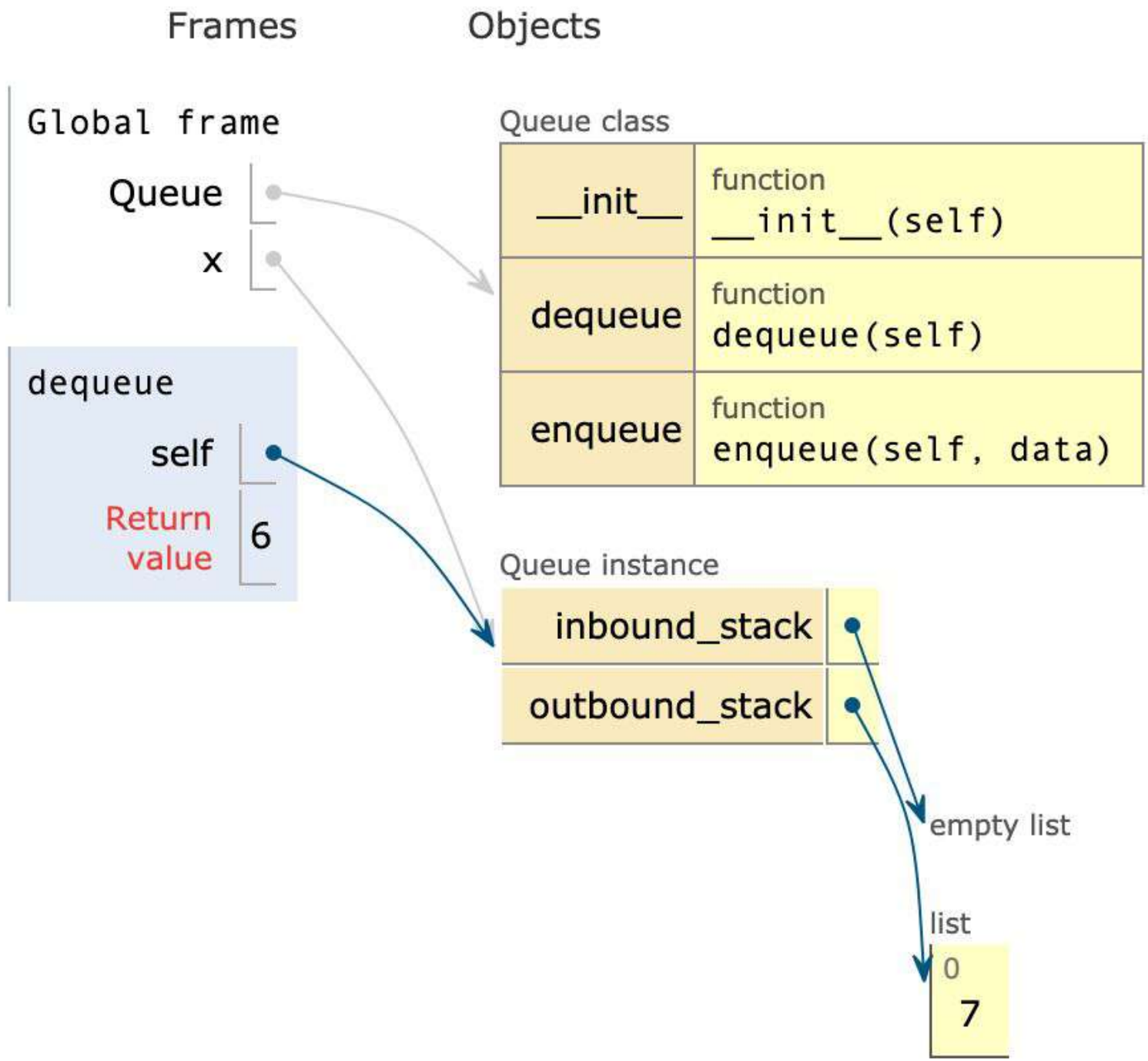


¿Qué hicimos?


Frames
Objects
Global frame
Queue
Queue class
__init__
function
__init__(self)
dequeue
function
dequeue(self)
enqueue
function
enqueue(self, data)


Frames
Objects
Global frame
Queue
x
dequeue
self
Queue class
__init__
function
__init__(self)
dequeue
function
dequeue(self)
enqueue
function
enqueue(self, data)
Queue instance
inbound_stack
outbound_stack
list
0
5
1
6
2
7
empty list


Frames
Objects
Global frame
Queue
x
dequeue
self
Return value
5
Queue class
__init__
function
__init__(self)
dequeue
function
dequeue(self)
enqueue
function
enqueue(self, data)
Queue instance
inbound_stack
outbound_stack
empty list
list
0
7
1
6

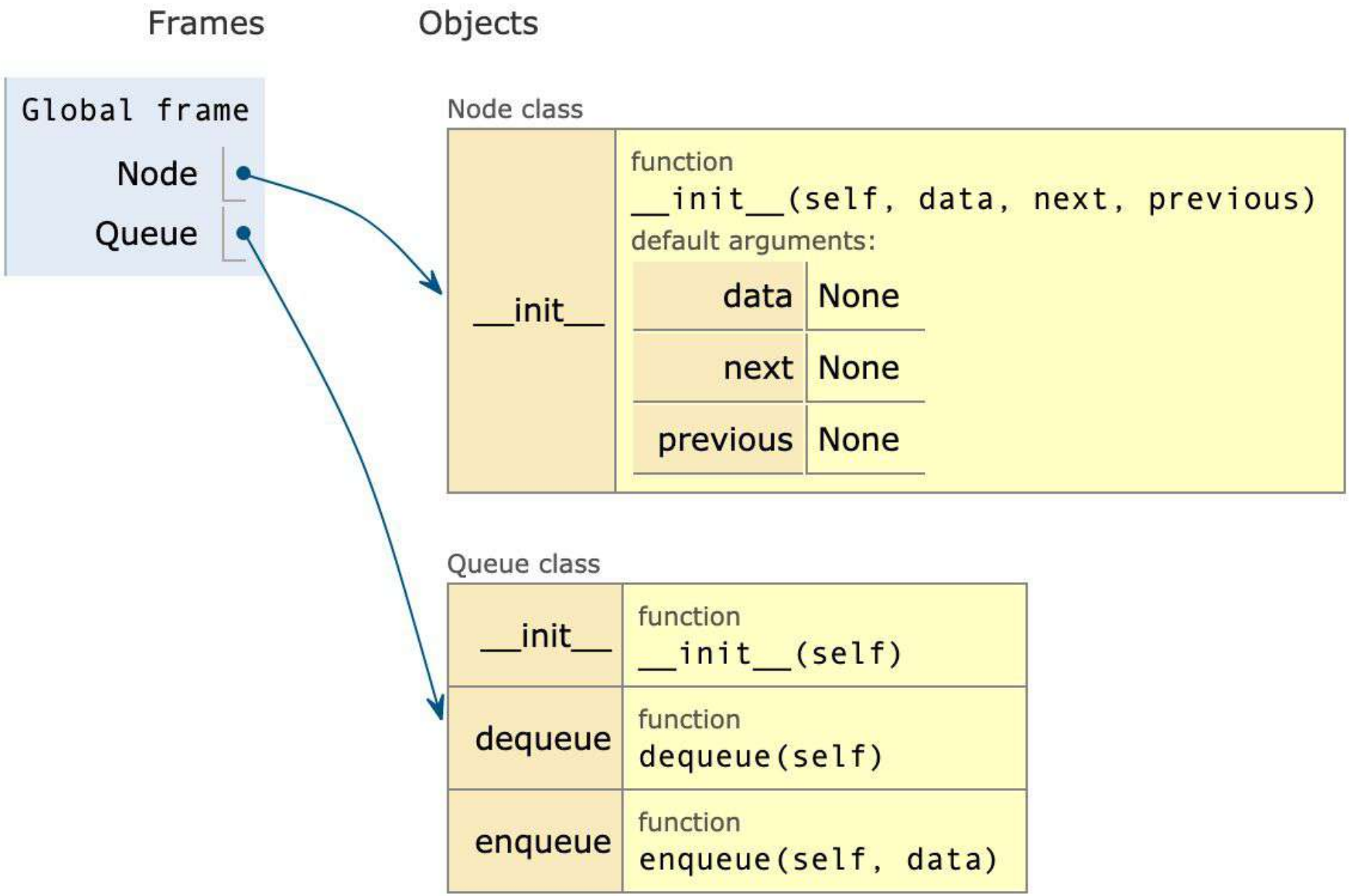

Frames
Objects
Global frame
Queue
x
dequeue
self
Return value
6
Queue class
__init__
function
__init__(self)
dequeue
function
dequeue(self)
enqueue
function
enqueue(self, data)
Queue instance
inbound_stack
outbound_stack
empty list
list
0
7

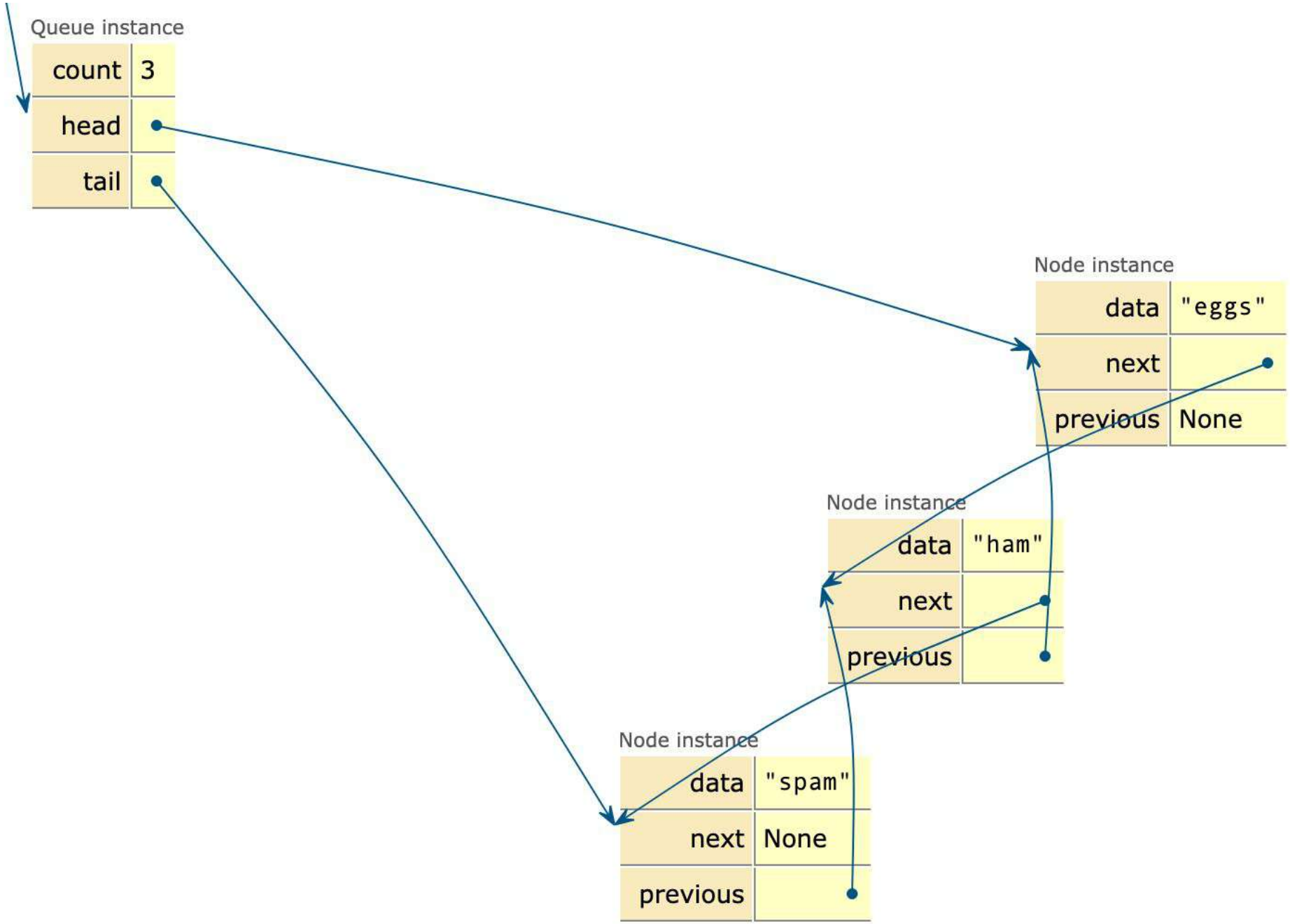
Queues basada nodos

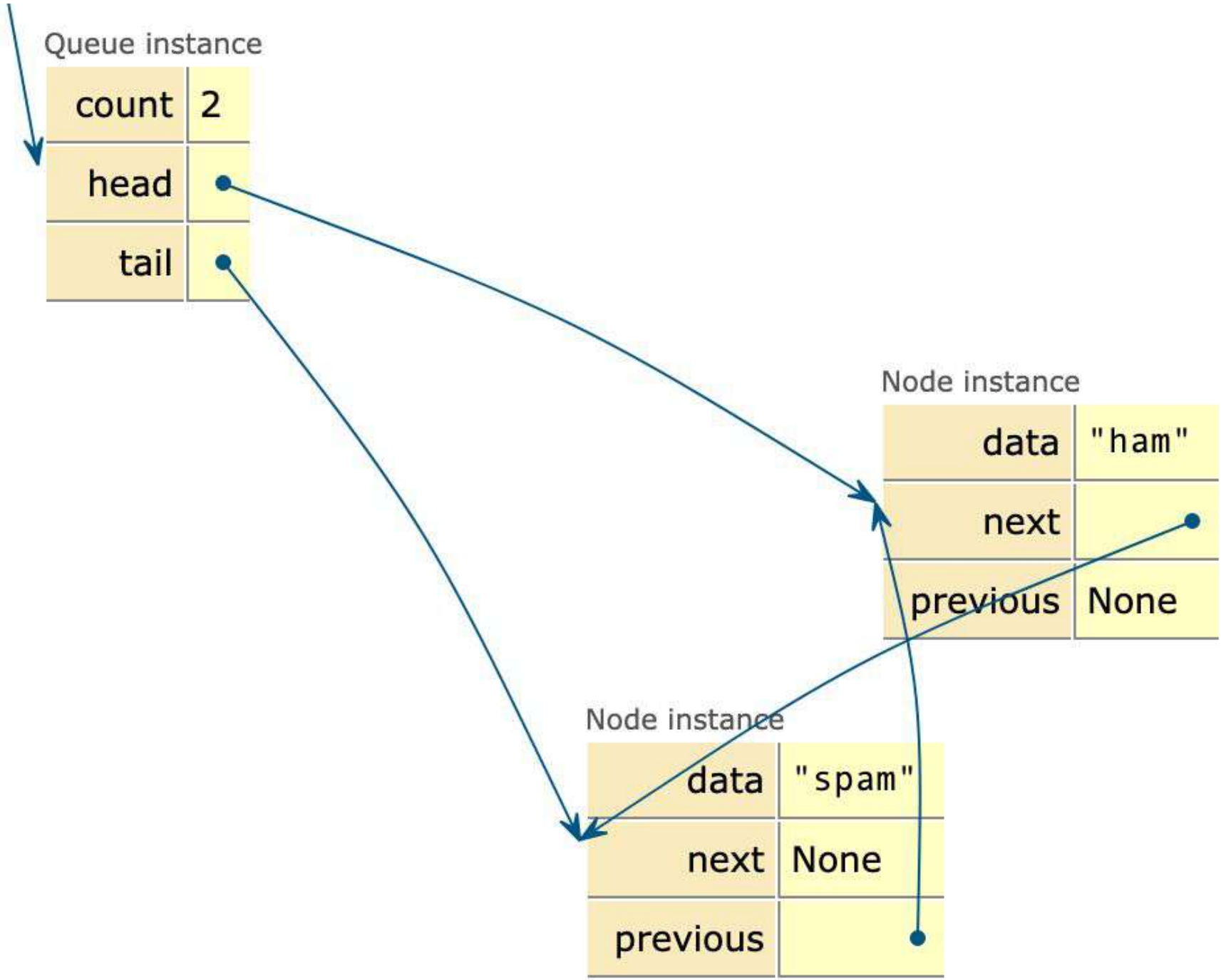
🐍 ¿Por qué y cómo?

- Agregar y eliminar elementos son O(1).
- Usando doubly linked list.

¿Qué hicimos?


Frames
Objects
Global frame
Node
Queue
Node class
__init__
function
__init__(self, data, next, previous)
default arguments:
data
None
next
None
previous
None
Queue class
__init__
function
__init__(self)
dequeue
function
dequeue(self)
enqueue
function
enqueue(self, data)


Queue instance
count
3
head
tail
Node instance
data
"eggs"
next
previous
None
Node instance
data
"ham"
next
previous
Node instance
data
"spam"
next
None
previous


Queue instance
count
2
head
tail
Node instance
data
"ham"
next
previous
None
Node instance
data
"spam"
next
None
previous

Reto: simulador de playlist musical

Restricciones

- Utiliza queues.
- Hay métodos para:
  - Añadir canciones.
  - Reproducirlas.
- Se reproducen las canciones en el orden añadido (FIFO).

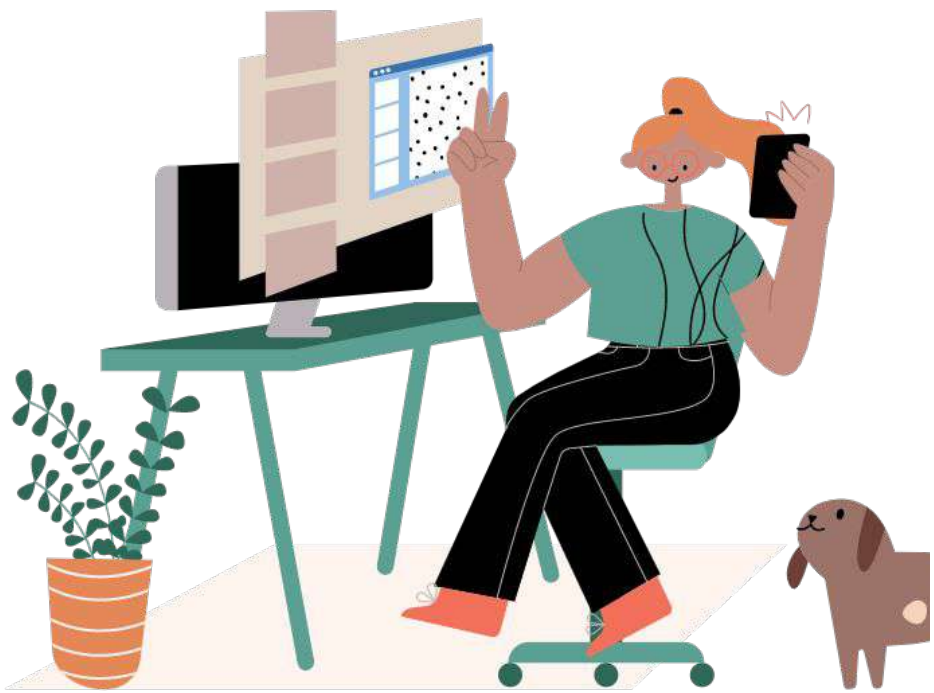
Más allá de las estructuras lineales

¿Qué has aprendido?

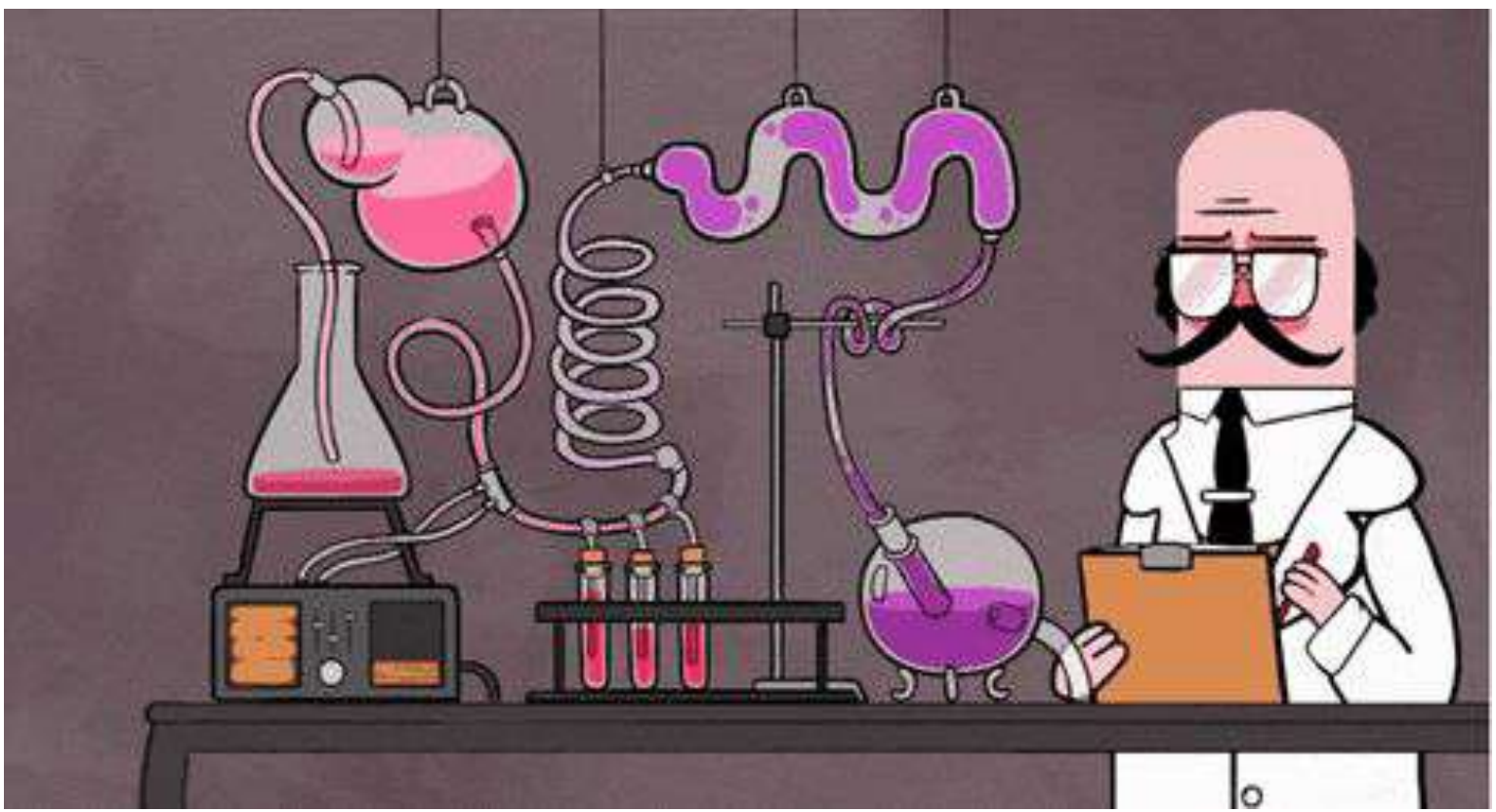
- Estructuras básicas:
  - Array
  - Node
- Linked list
- Stack
- Queue

¿Qué puedes hacer?

- Crear tus propias estructuras lineales.
- Optimizar tu código.
- Optimizar el uso de memoria.



¡Experimenta!



Felicidades
¡Nunca pares de aprender!