Curso de

POO y Algoritmos en Python

David Aroesti

Objetivos

- Entender cómo funciona la Programación Orientada a Objetos.
- Entender cómo medir la eficiencia temporal y espacial de nuestros algoritmos.
- Entender cómo y por qué graficar.
- Aprender a resolver problemas de búsqueda, ordenación y optimización.

Programación orientada a objetos

Tipos de datos abstractos, clases e instancias

Tipos de datos abstractos

- En Python todo es un objeto y tiene un tipo.
  - Representación de datos y formas de interactuar con ellos.

- Formas de interactuar con un objeto:
  - Creación
  - Manipulación
  - Destrucción

Tipos de datos abstractos

- Ventajas:
  - Decomposición
  - Abstracción
  - Encapsulación

```
# definición de clase

class <nombre_de_la_clase>(<super_clase>):

    def __init__(self, <params>):
        <expresion>

    def <nombre_del_metodo>(self, <params>):
        <expresion>
```

```python
# Definición
class Persona:

    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad

    def saluda(self, otra_persona):
        return f'Hola {otra_persona.nombre, me
llamo {self.nombre}.'


# Uso
>>> david = Persona('David', 35)
>>> erika = Persona('Erika', 32)

>>> david.saluda(erika)
'Hola Erika, me llamo David'
```

Instancias

Instancias

- Mientras que la clase es un molde, a los objetos creados se les conoce como instancias.
- Cuando se crea una instancia, se ejecuta el método `__init__`
- Todos los métodos de una clase reciben implícitamente como primer parámetro `self`

Instancias

- Los atributos de clase nos permiten:
  - Representar datos
  - Procedimientos para interactuar con los mismos (métodos)
  - Mecanismos para esconder la representación interna.

- Se accede a los atributos con la notación de punto.

- Puede tener atributos privados. Por convención comienzan con _

Decomposición

Decomposición

- Partir un problema en problemas más pequeños.
- Las clases permiten crear mayores abstracciones en forma de componentes.
- Cada clase se encarga de una parte del problema y el programa se vuelve más fácil de mantener

Abstracción

Abstracción

- Enfocarnos en la información relevante.
- Separar la información central de los detalles secundarios.
- Podemos utilizar variables y métodos (privados o públicos)

Encapsulación
y
getters and setters

Encapsulación

- Permite agrupar datos y su comportamiento.
- Controla el acceso a dichos datos.
- Previene modificaciones no autorizadas.

```
class CasillaDeVotacion:

    def __init__(self, identificador, pais)
        self._identificador = identificador
        self._pais = pais
        self._region = None

    @property
    def region(self):
        return self._region

    @region.setter
    def set_region(self, region):
        if region in self._pais:
            self._region = region

        raise ValueError(f'La region {region} no es valida en
{self._pais}')


>>> casilla = CasillaDeVotacion(123, ['Ciudad de Mexico', ‘Morelos’])
>>> casilla.region
None
>>> casilla.region = 'Ciudad de Mexico'
>>> casilla.region
'Ciudad de México'
```

Herencia

Herencia

- Permite modelar una jerarquía de clases.
- Permite compartir comportamiento común en la jerarquía.
- Al padre se le conoce como superclase y al hijo como subclase.

Polimorfismo

Polimorfismo

- La habilidad de tomar varias formas.
- En Python, nos permite cambiar el comportamiento de una superclase para adaptarlo a la subclase.

Introducción a la complejidad algorítmica

Introducción a la complejidad algorítmica

- ¿Por qué comparamos la eficiencia de un algoritmo?
- Complejidad temporal vs complejidad espacial
- Podemos definirla como T(n)

Aproximaciones

- Cronometrar el tiempo en el que corre un algoritmo.
- Contar los pasos con una medida abstracta de operación.
- Contar los pasos conforme nos aproximamos al infinito.

```python
def f(x):

    respuesta = 0

    for i in range(1000):
        respuesta += 1

    for i in range(x):
        respuesta += x

    for i in range(x):
        for j in range(x):
            respuesta += 1
            respuesta += 1

    return respuesta
```

Notación asintótica

Complejidad Algorítmica

Crecimiento asintótico

- No importan variaciones pequeñas.
- El enfoque se centra en lo que pasa conforme el tamaño del problema se acerca al infinito.
- Mejor de los casos, promedio, peor de los casos
- Big O
- Nada más importa el término de mayor tamaño

```python
# Ley de la suma

def f(n):
    for i in range(n):
        print(i)

    for i in range(n):
        print(i)

# O(n) + O(n) = O(n + n) = O(2n) = O(n)
```

```python
# Ley de la suma

def f(n):
    for i in range(n):
        print(i)

    for i in range(n * n):
        print(i)

# O(n) + O(n * n) = O(n + n²) = O(n²)
```

```python
# Ley de la multiplicación


def f(n):

    for i in range(n):

        for j in range(n):
            print(i, j)


# O(n) * O(n) = O(n * n) = O(n²)
```

```python
# Recursividad múltiple

def fibonacci(n):

    if n == 0 or n == 1:
        return 1

    return fibonacci(n - 1) + fibonacci(n - 2)


# O(2**n)
```

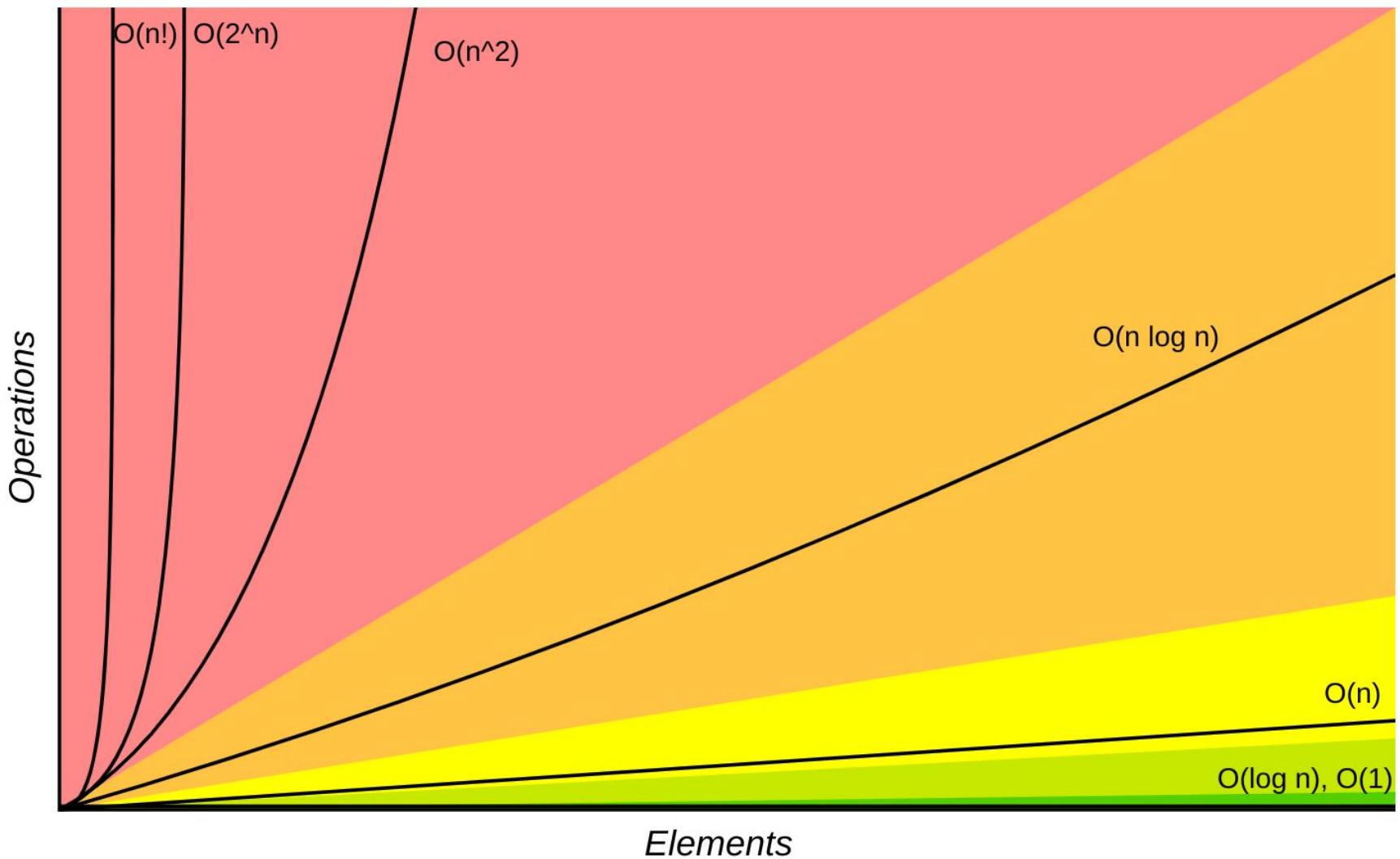
Clases de complejidad algorítmica

Complejidad Algorítmica

Clases de complejidad algorítmica

- O(1) Constante
- O(n) Lineal
- O(log n) Logarítmica
- O(n log n) log lineal
- O(n**2) Polinomial
- O(2**n) Exponencial

Clases de complejidad algorítmica



Clases de complejidad algorítmica

CLASS	n=10	= 100	= 1000	= 1000000
O(1)	1	1	1	1
O(log n)	1	2	3	6
O(n)	10	100	1000	1000000
O(n log n)	10	200	3000	6000000
O(n^2)	100	10000	1000000	1000000000000
O(2^n)	1024	1267650600228229401496703205376	1071508607186267320948425049060001810561404811705533607443750388370351051124936122493198378815695858127594672917553146825187145285692314043598457757469857480393456777482423098542107460506237114187795418215304647498358194126739876755916554394607706291457119647768654216766042983165262438683720566806937 6	Good luck!!

Búsqueda lineal

Algoritmos de búsqueda y ordenación

Búsqueda lineal

- Busca en todos los elementos de manera secuencial.
- ¿Cuál es el peor caso?

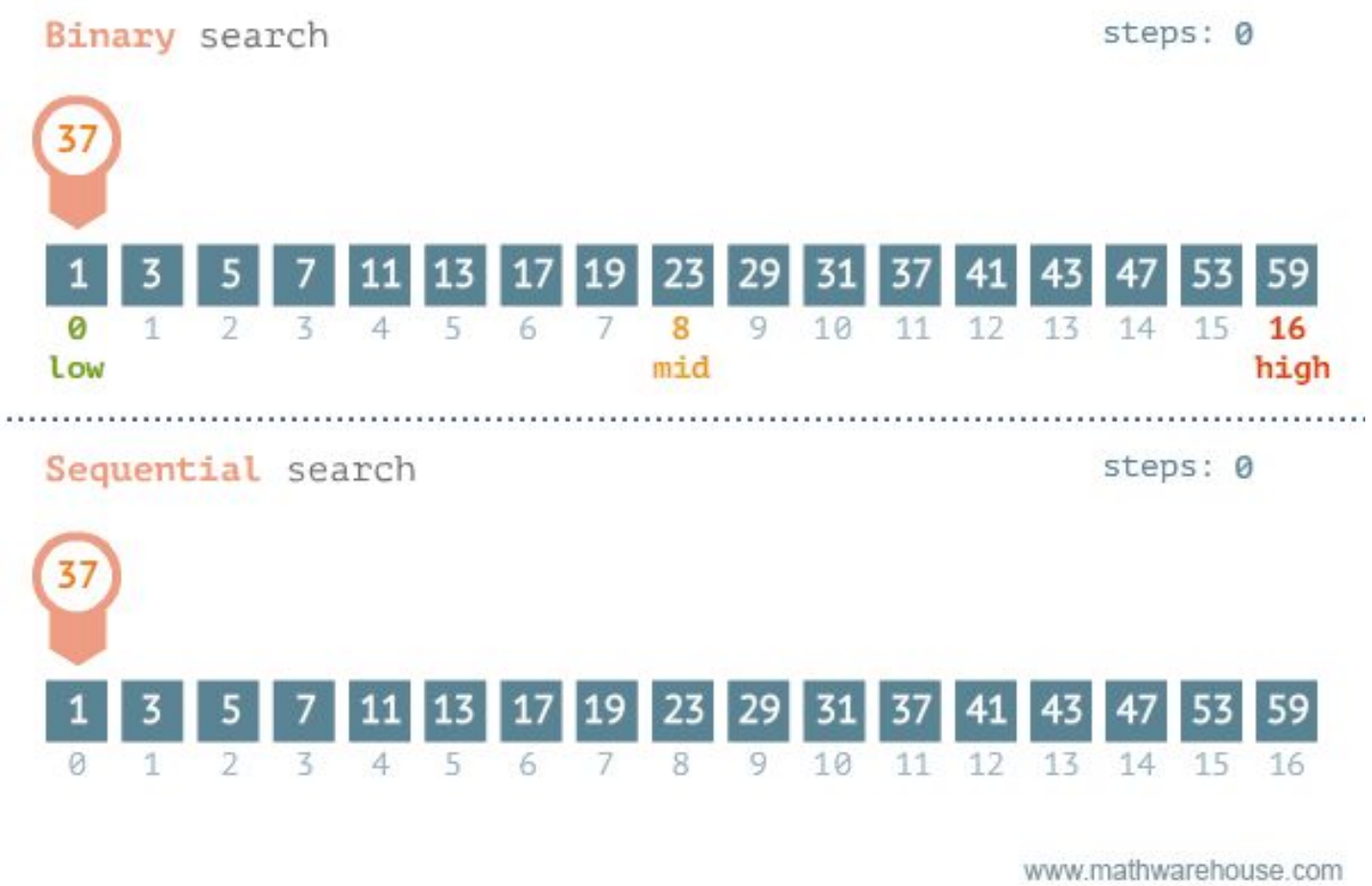
Búsqueda binaria

Algoritmos de búsqueda y ordenación

Búsqueda binaria

- Divide y conquista.
- El problema se divide en 2 en cada iteración.
- ¿Cuál es el peor caso?

Búsqueda binaria



Ordenamiento de burbuja

Algoritmos de búsqueda y ordenación

“

El ordenamiento de burbuja es un algoritmo que recorre repetidamente una lista que necesita ordenarse. Compara elementos adyacentes y los intercambia si están en el orden incorrecto. Este procedimiento se repite hasta que no se requieren más intercambios, lo que indica que la lista se encuentra ordenada.

”

Ordenamiento de burbuja

6 5 3 1 8 7 2 4

Ordenamiento por mezcla

Algoritmos de búsqueda y ordenación

“

El ordenamiento por mezcla es un algoritmo de divide y conquista. Primero divide una lista en partes iguales hasta que quedan sublistas de 1 o 0 elementos. Luego las recombina en forma ordenada.

”

Ordenamiento por mezcla

6 5 3 1 8 7 2 4

Ambientes virtuales

Ambientes virtuales

Ambientes virtuales

- Permiten aislar el ambiente para poder instalar diversas versiones de paquetes.
- A partir de python 3 se incluye en la librería estándar en el módulo venv.
- Ningún ingeniero profesional de Python trabaja sin ellos.

Pip

- Permite descargar paquetes de terceros para utilizar en nuestro programa.
- Permite compartir nuestros paquetes con terceros.
- Permite especificar la versión del paquete que necesitamos.

¿Por qué graficar?

Graficado

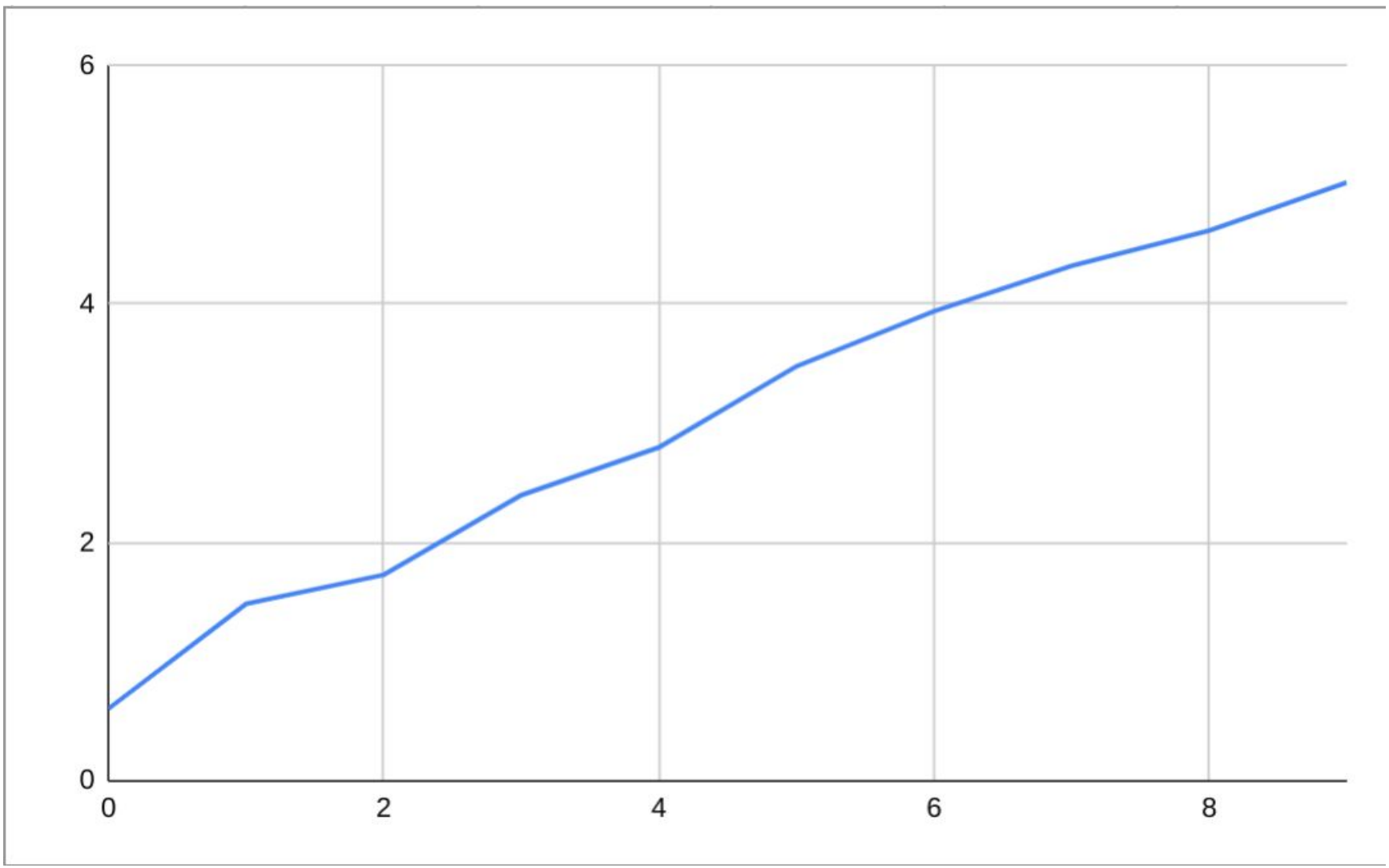
¿Por qué graficar?

- Reconocimiento de patrones
- Predicción de una serie
- Simplifica la interpretación y las conclusiones acerca de los datos

¿Por qué graficar?

0	0.6063801459
1	1.485049375
2	1.727423343
3	2.396435356
4	2.796497558
5	3.476608153
6	3.938803065
7	4.317980629
8	4.614371056
9	5.018067137

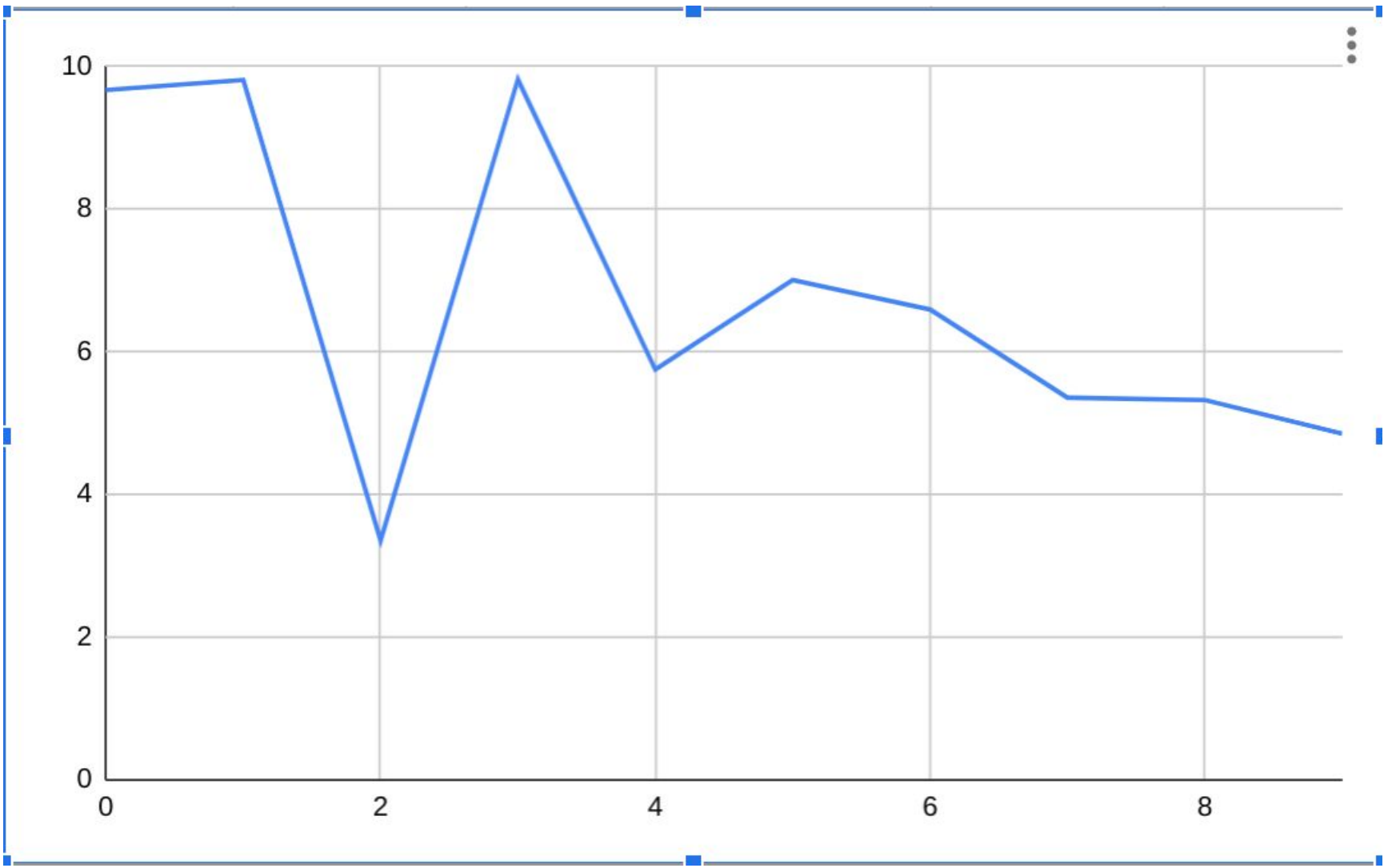
¿Por qué graficar?



¿Por qué graficar?

0	9.662053885
1	9.80444002
2	3.353063786
3	9.81227814
4	5.750058326
5	7.002114666
6	6.590261341
7	5.353824164
8	5.318413319
9	4.850503699

¿Por qué graficar?



Graficado simple

Graficado

Graficado simple

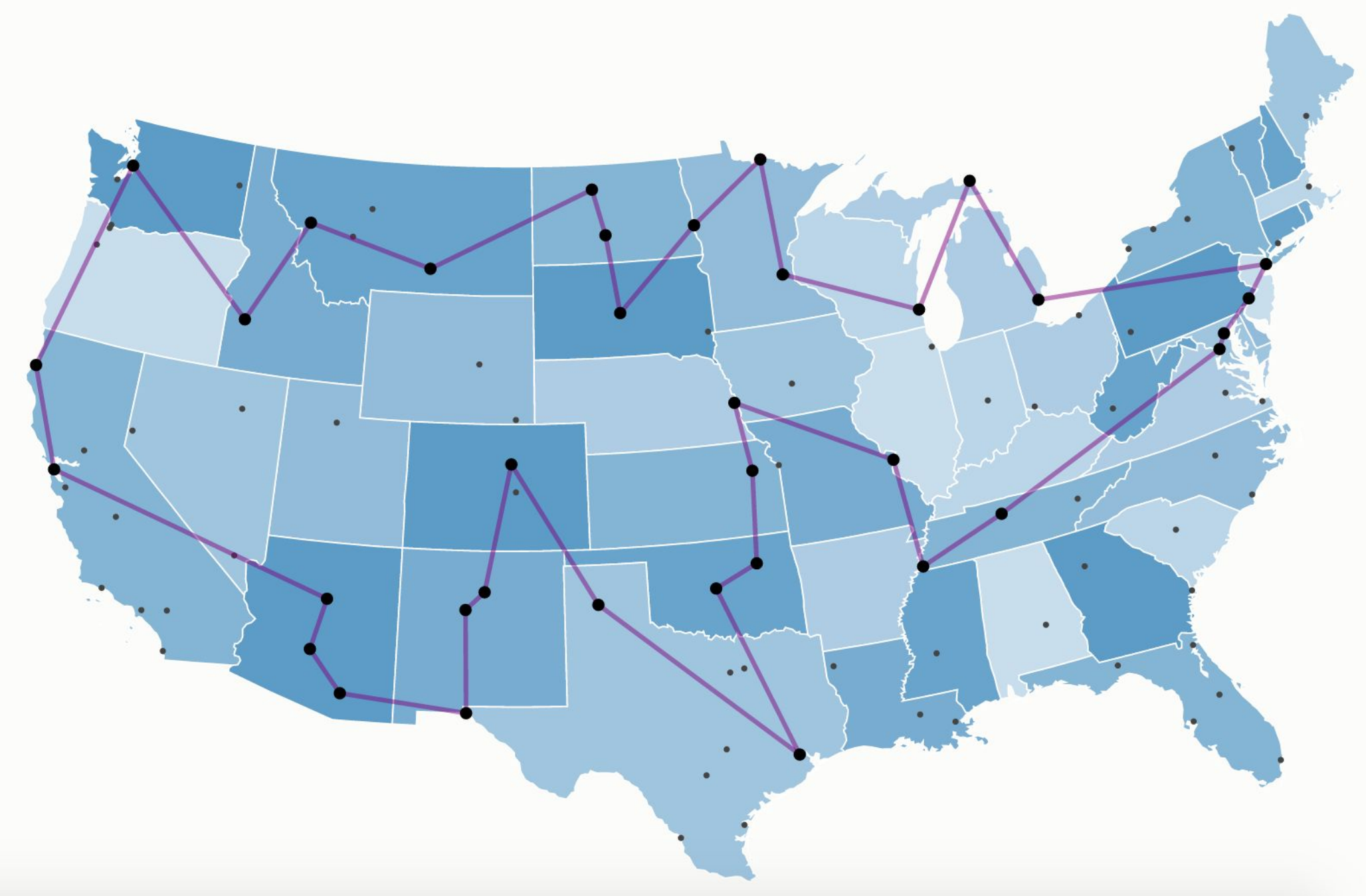
- Bokeh permite construir gráficas complejas de manera rápida y con comandos simples.
- Permite exportar a varios formatos como html, notebooks, imágenes, etc.
- Bokeh se puede utilizar en el servidor con Flask y Django.

Introducción a la optimización

Algoritmos de optimización

Introducción a la optimización

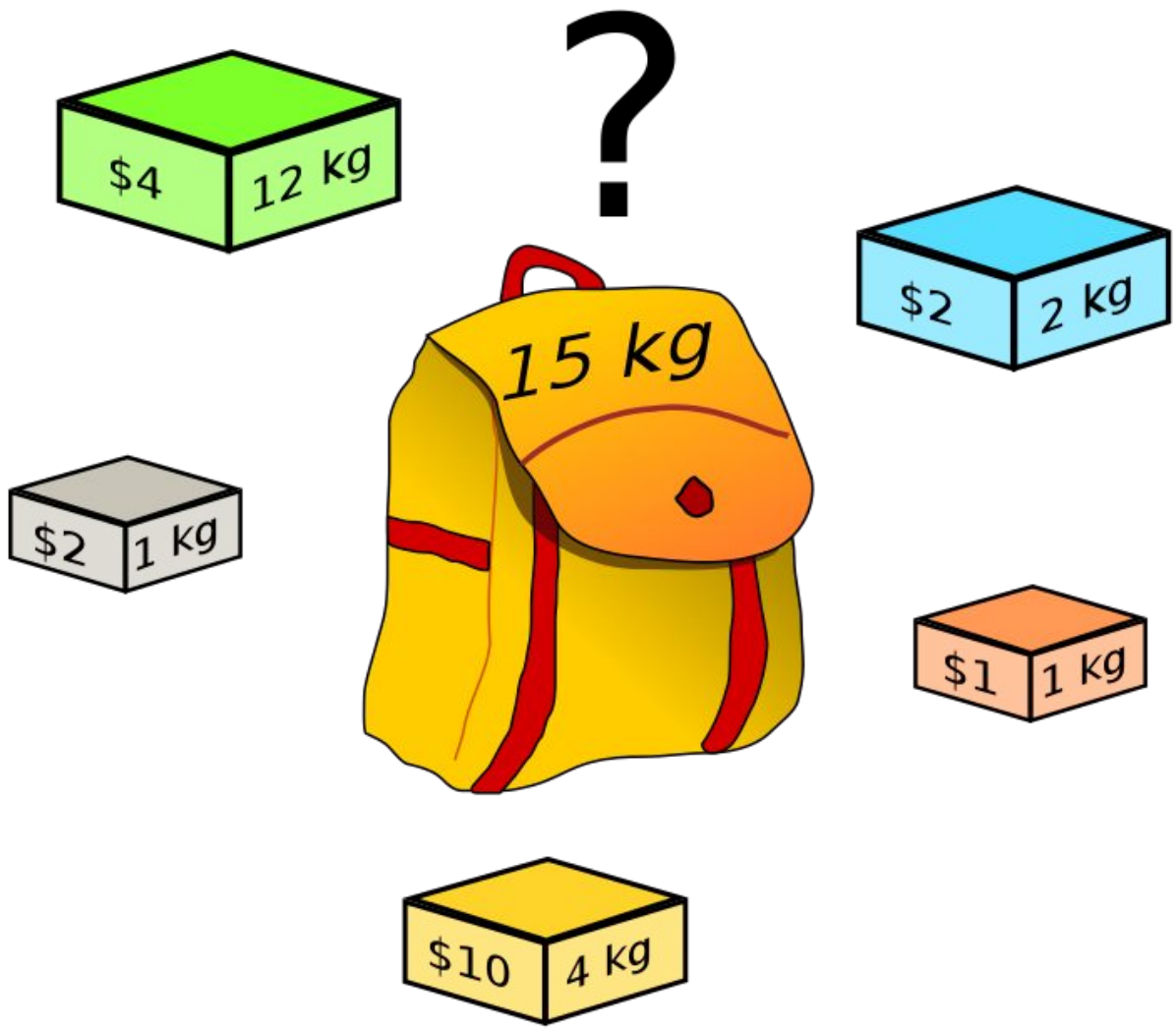
- El concepto de optimización permite resolver mucho problemas de manera computacional.
- Una función objetivo que debemos maximizar o minimizar.
- Una serie de limitantes que debemos respetar.



El problema del morral

Algoritmos de optimización

El problema del morral



Conclusiones

Conclusiones

Conclusiones

- Los tipos abstractos (clases) permiten crear programas poderosos que modelan al mundo.
- Podemos medir la eficiencia de diversos algoritmos.
- Las gráficas nos permiten encontrar patrones rápidamente.
- Optimización

Conclusiones

- Pensamiento computacional desarrollado:
  - Decomposición
  - Abstracción
  - Reconocimiento de patrones
  - Diseño de algoritmos