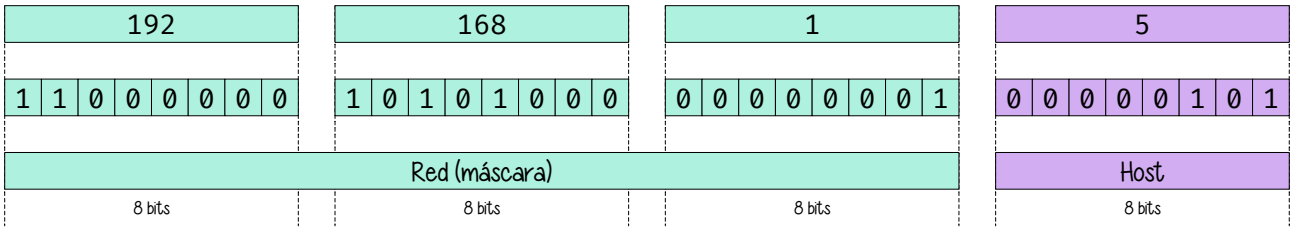
Redes

Este ejercicio gira en torno a la gestión de **redes de ordenadores** desarrollando un programa en **Python** mediante el paradigma de **programación orientada a objetos**.

1. Manipulación de IPs

La idea es simular un **sistema de direccionamiento IP** en redes de ordenadores de tal forma que se puedan realizar operaciones sobre ellas.

A continuación se presenta un ejemplo de representación de la IP `192.168.1.5/24`



2. Clase `Host`

Representa un *host* o dispositivo de red que dispone de una IP (y de una máscara de red).

Implementa, *al menos*, los siguientes **métodos**:

```
def __init__(self, *ip_octets: int, mask: int):
```

- Constructor de la clase.
- Habrá que crear los atributos `ip_octets` y `mask` a partir de los argumentos.
- Si se reciben más o menos de 4 octetos habrá que lanzar una excepción de tipo `IPAddressError` con el mensaje: Only 4 octets are allowed
- Si algunos de los octetos no está entre 0 y 255 habrá que lanzar una excepción de tipo `IPAddressError` con el mensaje: Octet is out of range
- Si el valor de la máscara no está entre 0 y 32 habrá que lanzar una excepción de tipo `IPAddressError` con el mensaje: Mask is out of range
- Ejemplo de llamada al constructor: `Host(192, 168, 1, 5, mask=24)` que representa al *host* con la IP `192.168.1.5/24`

 CC BY 4.0

```python
@classmethod
def build_from_sip(cls, sip: str, *, mask: int) -> Host:
```

- Se trata de un **método de clase**.
- Construye (y devuelve) un `Host` a partir de su IP en formato *string* (y su máscara).
- Nota mental: `sip` ⇔ *string IP*
- Haz uso del constructor definido previamente.
- El tratamiento de excepciones es el mismo que en el constructor de la clase (*no debes hacerlo sino delegarlo al propio constructor*).
- Ejemplo de llamada a este método: `Host.build_from_sip('192.168.1.5', mask=24)` que devolvería el *host* con la IP `192.168.1.5/24`

```python
@property
def ip(self) -> str:
```

- Devuelve la IP del *host* en formato cadena de texto.
- Ejemplo: `'192.168.1.5'`

```python
@property
def bip(self) -> str:
```

- Devuelve la IP del *host* en formato binario.
- Cuidado porque cada octeto debe tener 8 bits (incluyendo ceros a la izquierda si fuera necesario).
- Ejemplo: `'11000000101010000000000100000101'`

```python
@property
def addr_bmask(self) -> str:
```

- Devuelve la parte de la dirección IP que representa la máscara (en binario).
- Haz uso de la propiedad `bip` definida previamente.

```python
@property
def addr_bhost(self) -> str:
```

- Devuelve la parte de la dirección IP que representa el *host* (en binario).
- Haz uso de la propiedad `bip` definida previamente.

```python
@property
def has_network_addr(self) -> bool:
```

- Indica si la dirección IP del *host* (`self`) corresponde con la dirección de red.
- En una dirección de red, la parte de *host* tiene todos sus bits a 0. Ejemplo: `192.168.1.0`

CC BY 4.0

```
@property
def has_broadcast_addr(self) -> bool:
```

- Indica si la dirección IP del *host* (`self`) corresponde con la dirección de *broadcast*.
- En una dirección de *broadcast*, la parte de *host* tiene todos sus bits a 1. Ejemplo: `192.168.1.255`

```
@property
def nclass(self) -> str | None:
```

- Devuelve la clase de red a la que pertenece el *host*.
- Tabla de apoyo:

Clase	Rango del primer octeto
A	1 - 126
B	128 - 191
C	192 - 223
`None`	En otro caso

```
@property
def addr_host_size(self) -> int:
```

- Devuelve el número de bits que ocupa la parte de *host* (`self`) en la dirección.

```
@property
def num_hosts(self) -> int:
```

- Devuelve el número de *hosts* que puede haber en la red a la que pertenece el *host* (`self`).
- Para calcular la cantidad de *hosts* en una red, usa la fórmula $2^n - 2$ donde n corresponde a la cantidad de bits para la parte de *host* en la dirección de red.
- Haz uso de la propiedad `addr_host_size` definida previamente.

```
def ping(self, host: Host) -> bool:
```

- Indica si un *host* (`self`) puede hacer "ping" a otro *host* (`host`).
- Para que dos *hosts* puedan "verse" deben estar en la misma red.

```
def __repr__(self):
```

- Devuelve la representación del *host* (`self`) en formato `ip/máscara`
- Ejemplo: `'192.168.1.5/24'`

 CC BY 4.0

```python
@classmethod
def build_from_bip(cls, bip: str, mask: int) -> Host:
```

- Crea (y devuelve) un *host* a partir de una dirección IP (en binario) y una máscara.
- Nota mental: `bip` ⇔ *binary IP*
- Ejemplo: `bip='10010100101000111101010101110101'` y `mask=8` debería crear el *host*: `148.163.213.117/8`
- Si se pasan más de 32 bits habrá que lanzar una excepción de tipo `IPAddressError` con el mensaje: Binary address is too long

```python
def __iter__(self):
```

- Iterador del *host* (`self`).
- Implementa este método como **función generadora**.
- Debe devolver todos los *hosts* existentes en la propia red que define la IP del *host*.
- Haz uso del método `build_from_bip()` definido previamente.
- Por ejemplo, para `192.168.1.5/24`, habría que devolver los siguientes **objetos de tipo** `Host`:
  - `192.168.1.1/24`
  - `192.168.1.2/24`
  - `...`
  - `192.168.1.254/24`

```python
def __add__(self, other: Host) -> Host:
```

- Suma de dos objetos de tipo `Host`.
- El *host* resultante tendrá:
  - Como IP la suma de cada octeto correspondiente (primero con primero, segundo con segundo, ...). **Si dicha suma sobrepasa 255 se pondrá 255**.
  - Como máscara la suma de las dos máscaras. **Si dicha suma sobrepasa 32 se pondrá 32**.

3. Clase `IPAddressError`

Representa un error en el direccionamiento IP.

Implementa, *al menos*, los siguientes **métodos**:

 CC BY 4.0

```
def __init__(self, message: str = ''):
```

- Constructor de la clase.
- Si no pasamos ningún mensaje al constructor, el mensaje por defecto debe ser IP address is invalid
- Si pasamos un mensaje al constructor, el mensaje debe añadirse al que está por defecto. Por ejemplo si el mensaje que pasamos es *Something wrong* debería convertirse en IP address is invalid: Something wrong
- No es necesario crear el método `__str__()`. Se puede resolver todo desde el constructor (recuerda el concepto de *herencia*).

CC BY 4.0