Advanced Computer Networking (ACN)

Exercise 1 – Solution

Prof. Dr.-Ing. Georg Carle

Sebastian Gallenmüller, Max Helm, Benedikt Jaeger,
Marcel Kempf, Patrick Sattler, Johannes Zirngibl

Chair of Network Architectures and Services
School of Computation, Information, and Technology
Technical University of Munich

Outline

Announcements

Tutorial1 – Problem 0: Getting Access

Tutorial1 – Problem 1: Git Access

Tutorial1 – Problem 2: SSH and Virtual Machine (VM) Access

Tutorial1 – Problem 3: Jupyter Introduction

Tutorial1 – Problem 4: IPv6

Announcements

For questions and problems:

- Always use this mail address: `acn@net.in.tum.de`
- If you reply to a mail always use Reply All, usually results in a faster response

Tutorial

- Deadline for tutorial1 was 15 minutes ago
- If you haven't yet, commit and push your solution now

Tutorial1 – Problem 0: Getting Access

Demo

- Clone repository
- Boot VMs and log in
- Merge remote branch
- Copy repository to VM
- Build SSH tunnel and start Jupyter notebook
- SSH agent

Tutorial1 – Problem 1: Git Access

1 a)

Explain the differences between the Git commands `add`, `commit`, and `push`.

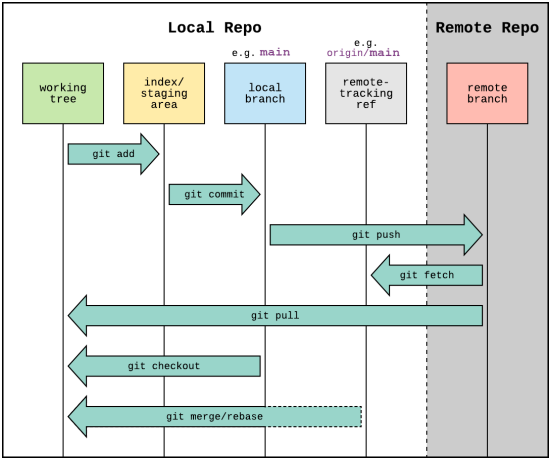
Tutorial1 – Problem 1: Git Access

1 a)

Explain the differences between the Git commands `add`, `commit`, and `push`.

According to `git man`:

- `git-add (1) Add file contents to the index.`
- `git-commit (1) Record changes to the repository.`
- `git-push (1) Update remote refs along with associated objects.`



https://www.reddit.com/r/git/comments/99ul9f/git_workflow_diagram_showcasing_the_role_of/

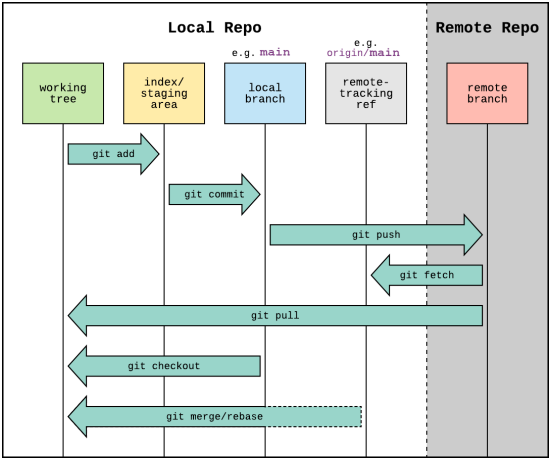
Tutorial1 – Problem 1: Git Access

1 a)

Explain the differences between the Git commands `add`, `commit`, and `push`.

According to `git man`:

- `git-add (1) Add file contents to the index.`
- `git-commit (1) Record changes to the repository.`
- `git-push (1) Update remote refs along with associated objects.`



https://www.reddit.com/r/git/comments/99ul9f/git_workflow_diagram_showcasing_the_role_of/

1 b)

Save your current changes to this Jupyter notebook. Add the file to a new commit and push to remote, then pull again. Execute the command git tag and paste the output here. Explain the meaning of the output.

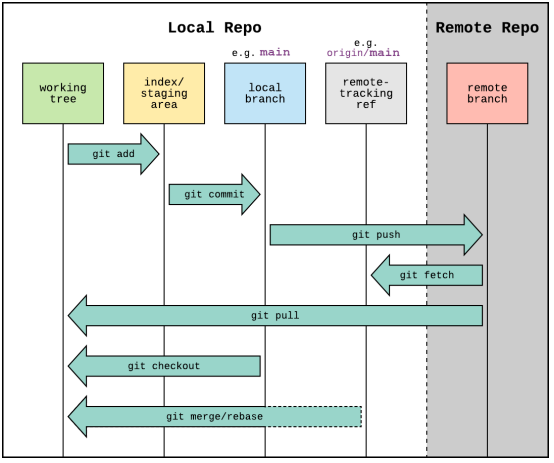
Tutorial1 – Problem 1: Git Access

1 a)

Explain the differences between the Git commands `add`, `commit`, and `push`.

According to `git man`:

- `git-add (1) Add file contents to the index.`
- `git-commit (1) Record changes to the repository.`
- `git-push (1) Update remote refs along with associated objects.`



https://www.reddit.com/r/git/comments/99ul9f/git_workflow_diagram_showcasing_the_role_of/

1 b)

Save your current changes to this Jupyter notebook. Add the file to a new commit and push to remote, then pull again. Execute the command git tag and paste the output here. Explain the meaning of the output.

`submission/1476987868` Unix timestamp:
`submission/1476989062` `1476987868 = Oct 20 2016 18:24:28`

1 c)

Create and push a new branch called `grades`. Paste the commands used to do so into your answer. Explain what happens.

1 c)

Create and push a new branch called `grades`. Paste the commands used to do so into your answer. Explain what happens.

- Push rule for branch names: `^main|grades$` → only `main` and `grades` branches can be pushed
- Protected branches: `grades` → you can only push to `main`, we can push to `grades`

Tutorial1 – Problem 2: SSH and Virtual Machine (VM) Access

2 a)

Explain what SSH is and what it is being used for.

Tutorial1 – Problem 2: SSH and Virtual Machine (VM) Access

2 a)

Explain what SSH is and what it is being used for.

> `The Secure Shell (SSH) is a protocol for secure remote login and other secure network services over an insecure network.` (RFC 4253)

- SSH = **S**ecure **SH**ell
- It is a protocol running on TCP port 22
- Provides encryption, host-based authentication and integrity protection
- Can be used for remote login and tunneling

Tutorial1 – Problem 2: SSH and Virtual Machine (VM) Access

2 a)

Explain what SSH is and what it is being used for.

> `The Secure Shell (SSH) is a protocol for secure remote login and other secure network services over an insecure network.` (RFC 4253)

- SSH = **S**ecure **SH**ell
- It is a protocol running on TCP port 22
- Provides encryption, host-based authentication and integrity protection
- Can be used for remote login and tunneling

2 b)

Explain the difference between public-key and password authentication as offered by SSH.

Tutorial1 – Problem 2: SSH and Virtual Machine (VM) Access

2 a)

Explain what SSH is and what it is being used for.

> `The Secure Shell (SSH) is a protocol for secure remote login and other secure network services over an insecure network.` (RFC 4253)

- SSH = **S**ecure **SH**ell
- It is a protocol running on TCP port 22
- Provides encryption, host-based authentication and integrity protection
- Can be used for remote login and tunneling

2 b)

Explain the difference between public-key and password authentication as offered by SSH.

- Password as shared secret between server and client (symmetric)
- Client uses (secret) private key to authenticate against public key on server

2 c)

SSH can be used to connect to your personal VM and to clone your personal git repository. You can connect to your personal VM using the command:

```
ssh -L localhost:1337:localhost:1337 root@svmNNNN.net.in.tum.de
```

where NNNN is your UID. Explain in detail what this command does.

2 c)

SSH can be used to connect to your personal VM and to clone your personal git repository. You can connect to your personal VM using the command:

```
ssh -L localhost:1337 : localhost:1337 root@svmNNNN.net.in.tum.de
```

where NNNN is your UID. Explain in detail what this command does.

The commands enables ssh tunneling from the local addess (IP and port) to the remote addess (IP and port) on the remote host .

2 c)

SSH can be used to connect to your personal VM and to clone your personal git repository. You can connect to your personal VM using the command:

```
ssh -L localhost:1337 : localhost:1337 root@svmNNNN.net.in.tum.de
```

where NNNN is your UID. Explain in detail what this command does.

The commands enables ssh tunneling from the local addess (IP and port) to the remote addess (IP and port) on the remote host .

2 d)

Connect to your virtual machine using SSH and execute the following three commands: `whoami`, `uname -a`, `pwd`. Paste the output of each of the three commands into your answer and explain what each command does.

2 c)

SSH can be used to connect to your personal VM and to clone your personal git repository. You can connect to your personal VM using the command:

```
ssh -L localhost:1337 : localhost:1337 root@svmNNNN.net.in.tum.de
```

where NNNN is your UID. Explain in detail what this command does.

The commands enables ssh tunneling from the local addess (IP and port) to the remote addess (IP and port) on the remote host .

2 d)

Connect to your virtual machine using SSH and execute the following three commands: `whoami`, `uname -a`, `pwd`. Paste the output of each of the three commands into your answer and explain what each command does.

- `whoami - display effective user id`
- `uname - display information about the system`
- `pwd - return working directory name`

Tutorial1 – Problem 3: Jupyter Introduction

3 a)

Jupyter notebooks consist of cells with different types, e. g., code and markdown.
Explain the differences between those two types and what they can be used for.

Tutorial1 – Problem 3: Jupyter Introduction

3 a)

Jupyter notebooks consist of cells with different types, e. g., code and markdown. Explain the differences between those two types and what they can be used for.

- A code cell supports the execution of Python code
- Only valid code can be executed
- Code cells also contain an execution number
- The markdown cell supports text with markdown syntax
- If needed HTML code can be used to format the text

Note: we look forward to nicely formatted answers :)

Tutorial1 – Problem 3: Jupyter Introduction

3 a)

Jupyter notebooks consist of cells with different types, e. g., code and markdown. Explain the differences between those two types and what they can be used for.

- A code cell supports the execution of Python code
- Only valid code can be executed
- Code cells also contain an execution number
- The markdown cell supports text with markdown syntax
- If needed HTML code can be used to format the text

Note: we look forward to nicely formatted answers :)

3 b)

Errors should never occur in your handed-in notebook. Fix this code by defining the `hello_world` variable, assigning it a value, and returning it.

Tutorial1 – Problem 3: Jupyter Introduction

3 a)

Jupyter notebooks consist of cells with different types, e. g., code and markdown. Explain the differences between those two types and what they can be used for.

- A code cell supports the execution of Python code
- Only valid code can be executed
- Code cells also contain an execution number
- The markdown cell supports text with markdown syntax
- If needed HTML code can be used to format the text

Note: we look forward to nicely formatted answers :)

3 b)

Errors should never occur in your handed-in notebook. Fix this code by defining the `hello_world` variable, assigning it a value, and returning it.

```
def hello_world_text():
        # begin insert code
        hello_world = "Hello World!"
        return hello_world
        # end insert code
        return None
print(hello_world_text())

> Hello World!
```

3 d)

Code cells also allow to execute shell commands. These are executed as the user who started the Jupyter server. On your virtual machine this is root. Shell commands can be executed by prefixing them with the '!' character.

3 d)

Code cells also allow to execute shell commands. These are executed as the user who started the Jupyter server. On your virtual machine this is root. Shell commands can be executed by prefixing them with the '!' character.

```
!pwd  # the path where jupyter has been started
!echo This user is executing the commands: $USER
!ping -c 1 net.in.tum.de

> /Users/sattler/acn/exercise/2021
> This user is executing the commands: sattler
> PING net.in.tum.de (131.159.15.24): 56 data bytes
> 64 bytes from 131.159.15.24: icmp_seq=0 ttl=63 time=0.561 ms

> --- net.in.tum.de ping statistics ---
> 1 packets transmitted, 1 packets received, 0.0% packet loss
> round-trip min/avg/max/stddev = 0.561/0.561/0.561/0.000 ms
```

3 d)

Code cells also allow to execute shell commands. These are executed as the user who started the Jupyter server. On your virtual machine this is root. Shell commands can be executed by prefixing them with the '!' character.

```
!pwd  # the path where jupyter has been started
!echo This user is executing the commands: $USER
!ping -c 1 net.in.tum.de

> /Users/sattler/acn/exercise/2021
> This user is executing the commands: sattler
> PING net.in.tum.de (131.159.15.24): 56 data bytes
> 64 bytes from 131.159.15.24: icmp_seq=0 ttl=63 time=0.561 ms

> --- net.in.tum.de ping statistics ---
> 1 packets transmitted, 1 packets received, 0.0% packet loss
> round-trip min/avg/max/stddev = 0.561/0.561/0.561/0.000 ms
```

3 e)

In some exercise sheets we will use such shell commands to install missing Python modules.

Tutorial1 – Problem 3: Jupyter Introduction

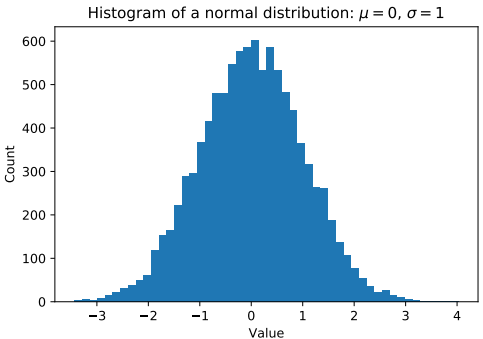
3 f)

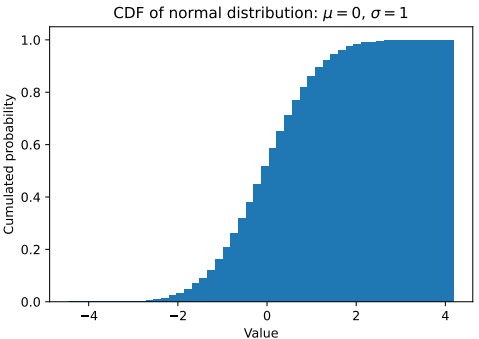
Use the data from the previous cell to produce a CDF.

3 f)

Use the data from the previous cell to produce a CDF.

CDF = Cumulative Distribution Function






```
ax.hist(x, num_bins, cumulative=True, density=True)
```

Tutorial1 – Problem 4: IPv6

4 a)

Write a function `convert_ipv6`.

- remove leading zeros for each byte-pair (do not remove trailing ones)
- the longest series of consecutive `0`s can be merged with `::`

4 a)

Write a function `convert_ipv6`.

- remove leading zeros for each byte-pair (do not remove trailing ones)
- the longest series of consecutive `0`s can be merged with `::`

```
0000:0000:0000:0000:0000:0000:0000:0000

0000:0000:0000:0000:0000:0000:0000:0001

ff01:0000:0000:0000:0000:0000:0000:0001

fe80:0000:0000:0000:4f21:13ff:fea1:dee2

fe80:0000:0000:0000:1f00:0000:0001:ede6

2001:4ca0:2001:3a40:e114:90fe:3862:444f
```

4 a)

Write a function `convert_ipv6`.

- remove leading zeros for each byte-pair (do not remove trailing ones) ✓
- the longest series of consecutive `0`s can be merged with `::` ✓

`0000:0000:0000:0000:0000:0000:0000:0000`	`::`	(unspecified address)
`0000:0000:0000:0000:0000:0000:0000:0001`		
`ff01:0000:0000:0000:0000:0000:0000:0001`		
`fe80:0000:0000:0000:4f21:13ff:fea1:dee2`		
`fe80:0000:0000:0000:1f00:0000:0001:ede6`		
`2001:4ca0:2001:3a40:e114:90fe:3862:444f`		

4 a)

Write a function `convert_ipv6`.

- remove leading zeros for each byte-pair (do not remove trailing ones) ✓
- the longest series of consecutive `0`s can be merged with `::` ✓

`0000:0000:0000:0000:0000:0000:0000:0000`	`::`	(unspecified address)
`0000:0000:0000:0000:0000:0000:0000:0001`	`::1`	(localhost)
`ff01:0000:0000:0000:0000:0000:0000:0001`		
`fe80:0000:0000:0000:4f21:13ff:fea1:dee2`		
`fe80:0000:0000:0000:1f00:0000:0001:ede6`		
`2001:4ca0:2001:3a40:e114:90fe:3862:444f`		

4 a)

Write a function `convert_ipv6`.

- remove leading zeros for each byte-pair (do not remove trailing ones) ✓
- the longest series of consecutive `0`s can be merged with `::` ✓

`0000:0000:0000:0000:0000:0000:0000:0000`	`::`	(unspecified address)
`0000:0000:0000:0000:0000:0000:0000:0001`	`::1`	(localhost)
`ff01:0000:0000:0000:0000:0000:0000:0001`	`ff01::1`	(all nodes multicast address)
`fe80:0000:0000:0000:4f21:13ff:fea1:dee2`		
`fe80:0000:0000:0000:1f00:0000:0001:ede6`		
`2001:4ca0:2001:3a40:e114:90fe:3862:444f`		

4 a)

Write a function `convert_ipv6`.

- remove leading zeros for each byte-pair (do not remove trailing ones) ✓
- the longest series of consecutive `0`s can be merged with `::` ✓

`0000:0000:0000:0000:0000:0000:0000:0000`	`::`	(unspecified address)
`0000:0000:0000:0000:0000:0000:0000:0001`	`::1`	(localhost)
`ff01:0000:0000:0000:0000:0000:0000:0001`	`ff01::1`	(all nodes multicast address)
`fe80:0000:0000:0000:4f21:13ff:fea1:dee2`	`fe80::4f21:13ff:fea1:dee2`	
`fe80:0000:0000:0000:1f00:0000:0001:ede6`		
`2001:4ca0:2001:3a40:e114:90fe:3862:444f`		

4 a)

Write a function `convert_ipv6`.

- remove leading zeros for each byte-pair (do not remove trailing ones) ✓
- the longest series of consecutive `0`s can be merged with `::` ✓

`0000:0000:0000:0000:0000:0000:0000:0000`	`::`	(unspecified address)
`0000:0000:0000:0000:0000:0000:0000:0001`	`::1`	(localhost)
`ff01:0000:0000:0000:0000:0000:0000:0001`	`ff01::1`	(all nodes multicast address)
`fe80:0000:0000:0000:4f21:13ff:fea1:dee2`	`fe80::4f21:13ff:fea1:dee2`	
`fe80:0000:0000:0000:1f00:0000:0001:ede6`	`fe80::1f00:0:1:ede6`	
`2001:4ca0:2001:3a40:e114:90fe:3862:444f`		

4 a)

Write a function `convert_ipv6`.

- remove leading zeros for each byte-pair (do not remove trailing ones) ✓
- the longest series of consecutive `0`s can be merged with `::` ✓

`0000:0000:0000:0000:0000:0000:0000:0000`	`::`	(unspecified address)
`0000:0000:0000:0000:0000:0000:0000:0001`	`::1`	(localhost)
`ff01:0000:0000:0000:0000:0000:0000:0001`	`ff01::1`	(all nodes multicast address)
`fe80:0000:0000:0000:4f21:13ff:fea1:dee2`	`fe80::4f21:13ff:fea1:dee2`	
`fe80:0000:0000:0000:1f00:0000:0001:ede6`	`fe80::1f00:0:1:ede6`	
`2001:4ca0:2001:3a40:e114:90fe:3862:444f`	`2001:4ca0:2001:3a40:e114:90fe:3862:444f`	

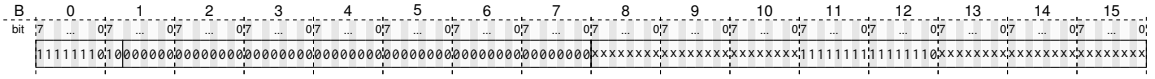
4 b)

Write a function `generate_link_local`.

Tutorial1 – Problem 4: IPv6

4 b)

Write a function `generate_link_local`.

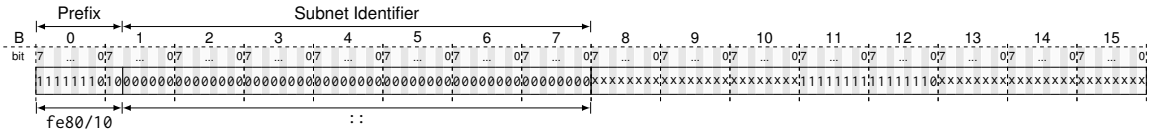


Link-Local Address as specified in RFC4291:

Tutorial1 – Problem 4: IPv6

4 b)

Write a function `generate_link_local`.



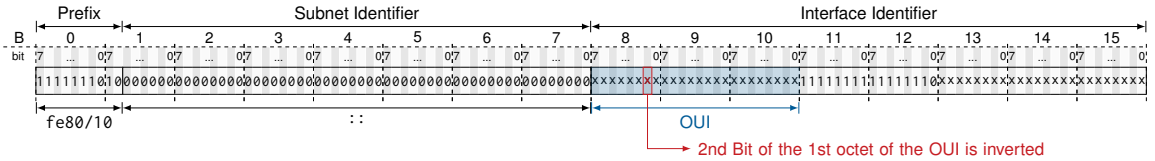
Link-Local Address as specified in RFC4291:

- First 10 bit: `1111 1110 10` (`0xfe80`)
- Remainder of the first 8 B set to `0`

Tutorial1 – Problem 4: IPv6

4 b)

Write a function `generate_link_local`.



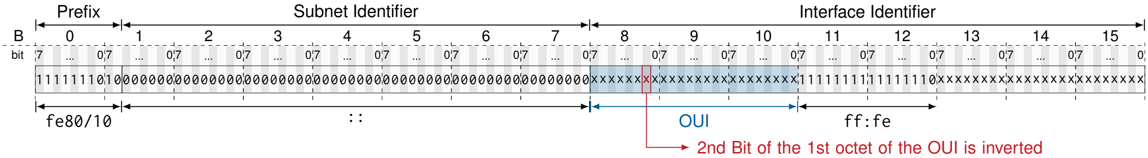
Link-Local Address as specified in RFC4291:

- First 10 bit: `1111 1110 10` (`0xfe80`)
- Remainder of the first 8 B set to `0`
- OUI of the MAC address (first 3 B)
  → flip 2nd Bit of first Byte

- The interface identifier is built according to the modified EUI-64 format

4 b)

Write a function `generate_link_local`.



Link-Local Address as specified in RFC4291:

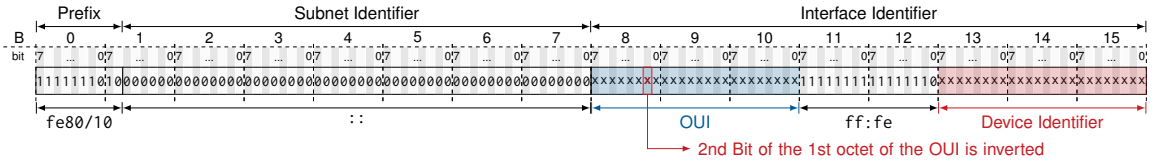
- First 10 bit: `1111 1110 10` (`0xfe80`)
- Remainder of the first 8 B set to `0`
- OUI of the MAC address (first 3 B) → flip 2nd Bit of first Byte
- `1111 1111 1111 1110` (`0xfffe`)

- The interface identifier is built according to the modified EUI-64 format

Tutorial1 – Problem 4: IPv6

4 b)

Write a function `generate_link_local`.



Link-Local Address as specified in RFC4291:

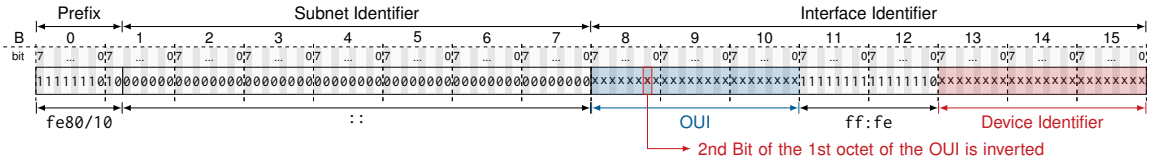
- First 10 bit: `1111 1110 10` (`0xfe80`)
- Remainder of the first 8 B set to `0`
- OUI of the MAC address (first 3 B)
  → flip 2nd Bit of first Byte
- `1111 1111 1111 1110` (`0xfffe`)
- Device Identifier of the MAC address (last 3 B)

- The interface identifier is built according to the modified EUI-64 format

Tutorial1 – Problem 4: IPv6

4 b)

Write a function `generate_link_local`.



Link-Local Address as specified in RFC4291:

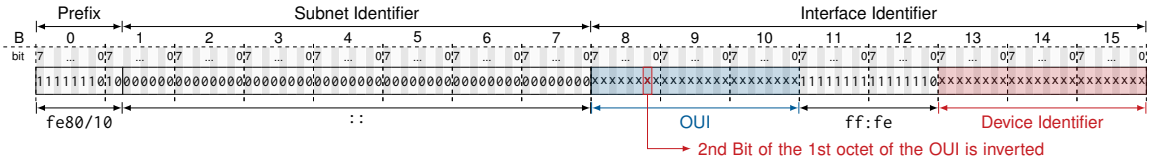
- First 10 bit: `1111 1110 10` (`0xfe80`)
- Remainder of the first 8 B set to `0`
- OUI of the MAC address (first 3 B)
  → flip 2nd Bit of first Byte
- `1111 1111 1111 1110` (`0xfffe`)
- Device Identifier of the MAC address (last 3 B)
- Example:
  `ether 7a:19:0e:68:46:d6`
  `inet6 fe80::7819:eff:fe68:46d6/64`

- The interface identifier is built according to the modified EUI-64 format

Tutorial1 – Problem 4: IPv6

4 b)

Write a function `generate_link_local`.



Link-Local Address as specified in RFC4291:

- First 10 bit: `1111 1110 10` (`0xfe80`)
- Remainder of the first 8 B set to `0`
- OUI of the MAC address (first 3 B)
  → flip 2nd Bit of first Byte
- `1111 1111 1111 1110` (`0xfffe`)
- Device Identifier of the MAC address (last 3 B)
- Example:
  `ether 7a:19:0e:68:46:d6`
  `inet6 fe80::7819:eff:fe68:46d6/64`

Implementation Details:

- Create empty `bytearray`:
  `var = bytearray(16)`
- Set slices of the array:
  `var[0:2] = b'\x12\x34'`
- Flip single bits:
  `mac[0] ^= 0x02`

Tutorial1 – Problem 4: IPv6
The Universal/Local Bit

- The second-least-significant bit of the first octet in a MAC address
- 0 indicates universally administered address

Tutorial1 – Problem 4: IPv6
The Universal/Local Bit

- The second-least-significant bit of the first octet in a MAC address
- 0 indicates universally administered address

Why is it inverted?

Tutorial1 – Problem 4: IPv6
The Universal/Local Bit

- The second-least-significant bit of the first octet in a MAC address
- 0 indicates universally administered address

Why is it inverted?

RCF 4291 Section 2.5.1

> *make it easy for system administrators to hand configure non-global identifiers when hardware tokens are not available*

- Else the first bit would need to be set
- Impossible to use simple interface identifiers like ::1

4 c)

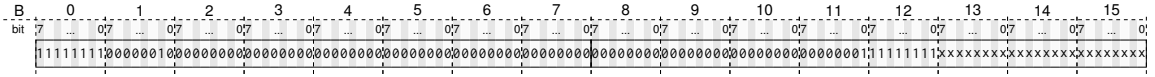
Write two functions:

- `compute_solicited_node_multicast`
- `compute_multicast_mac`

4 c)

Write two functions:

- `compute_solicited_node_multicast`
- `compute_multicast_mac`



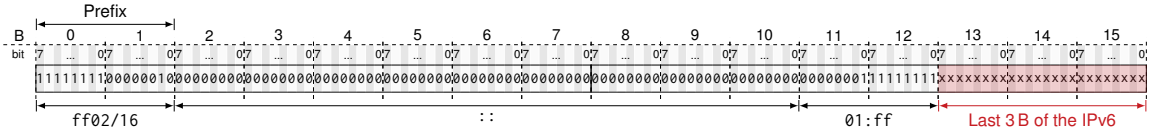
Solicited Node Multicast Address:

- From RFC4291: `ff02:0:0:0:0:1:ffxx:xxxx`, with `xx:xxxx` being the last 3 B of the node's IPv6 address

4 c)

Write two functions:

- `compute_solicited_node_multicast`
- `compute_multicast_mac`



Solicited Node Multicast Address:

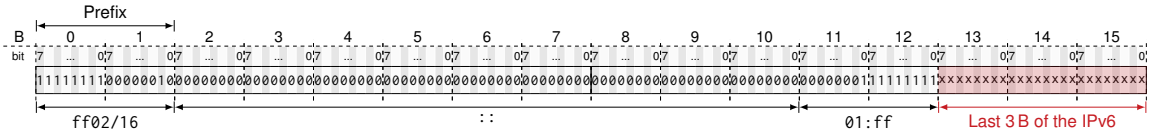
- From RFC4291: `ff02:0:0:0:0:1:ffxx:xxxx`, with `xx:xxxx` being the last 3 B of the node's IPv6 address
- Example: `2001:4ca0:2001:40:e114:90fe:3862:554f` → `ff02::1:ff62:554f`

Tutorial1 – Problem 4: IPv6

4 c)

Write two functions:

- `compute_solicited_node_multicast`
- `compute_multicast_mac`



Solicited Node Multicast Address:

- From RFC4291: `ff02:0:0:0:0:1:ffxx:xxxx`, with `xx:xxxx` being the last 3 B of the node's IPv6 address
- Example: `2001:4ca0:2001:40:e114:90fe:3862:554f` → `ff02::1:ff62:554f`

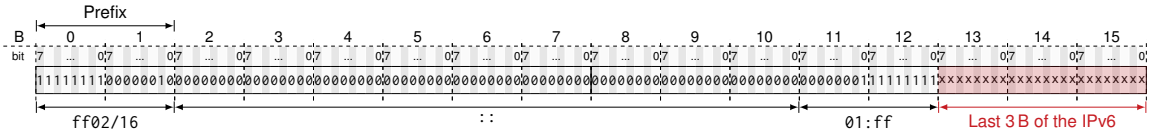
Multicast MAC:

Tutorial1 – Problem 4: IPv6

4 c)

Write two functions:

- `compute_solicited_node_multicast`
- `compute_multicast_mac`



Solicited Node Multicast Address:

- From RFC4291: `ff02:0:0:0:0:1:ffxx:xxxx`, with `xx:xxxx` being the last 3 B of the node's IPv6 address
- Example: `2001:4ca0:2001:40:e114:90fe:3862:554f` → `ff02::1:ff62:554f`

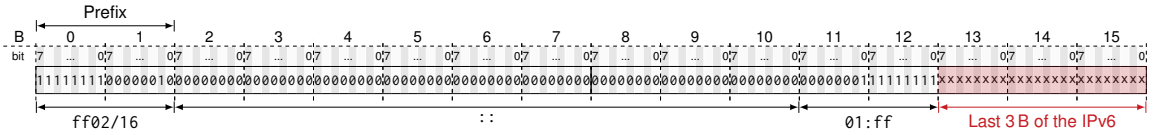
Multicast MAC:

- From RFC2464: `33:33:xx:xx:xx:xx`, with `xx:xx:xx:xx` being the last 4 B of the multicast IPv6 address

4 c)

Write two functions:

- `compute_solicited_node_multicast`
- `compute_multicast_mac`



Solicited Node Multicast Address:

- From RFC4291: `ff02:0:0:0:0:1:ffxx:xxxx`, with `xx:xxxx` being the last 3 B of the node's IPv6 address
- Example: `2001:4ca0:2001:40:e114:90fe:3862:554f` → `ff02::1:ff62:554f`

Multicast MAC:

- From RFC2464: `33:33:xx:xx:xx:xx`, with `xx:xx:xx:xx` being the last 4 B of the multicast IPv6 address
- Example: `ff02::1:ff62:554f` → `33:33:ff:62:55:4f`

4 d)

Write a function `count_ones` (only the last 64bits).

4 d)

Write a function `count_ones` (only the last 64bits).

```
811b : d503 : 8064 : b1ad : 59ba : 65d1 : df7e : 6720
```

4 d)

Write a function `count_ones` (only the last 64bits).

```
811b : d503 : 8064 : b1ad : 59ba : 65d1 : df7e : 6720

     :      :      :      :0101100110111010:0110010111010001:1101111101111110:0110011100100000
```

4 d)

Write a function `count_ones` (only the last 64bits).

```
811b : d503 : 8064 : b1ad : 59ba : 65d1 : df7e : 6720

     :      :      :      :0101100110111010:0110010111010001:1101111101111110:0110011100100000

                                  9              8             13             6
```

4 d)

Write a function `count_ones` (only the last 64bits).

```
811b : d503 : 8064 : b1ad : 59ba : 65d1 : df7e : 6720

                :      :      :0101100110111010:0110010111010001:1101111101111110:0110011100100000

                                                9          8         13         6
```

Implementation Details:

- Work on `bytearray` or `strings`?
- `'01010111'.count('1') = 5`
- `bin(int.from_bytes(bytearray(b'\xff\xfe'), byteorder='big', signed=False)).count('1')`

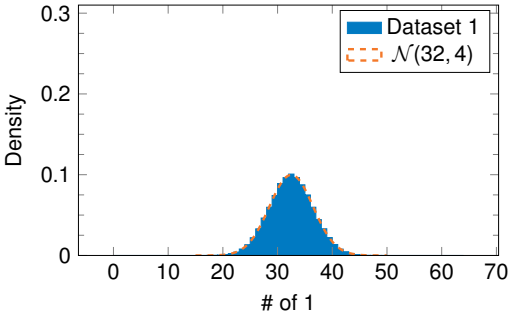
Tutorial1 – Problem 4: IPv6

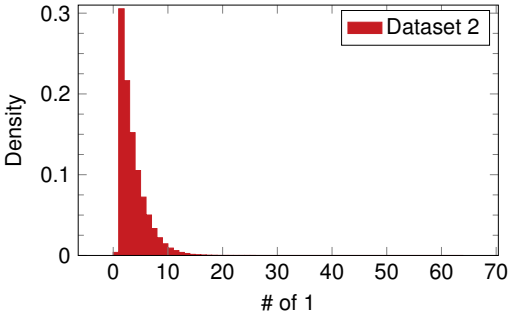
4 e)

Explain how the addresses in the two datasets differ.
Give a reason for the differences and what kind of addresses are most likely contained in each dataset.

4 e)

Explain how the addresses in the two datasets differ.
Give a reason for the differences and what kind of addresses are most likely contained in each dataset.






```
6519:1476:c881:1908:d668:f0b2:1e02:7728
811b:d503:8064:b1ad:59ba:65d1:df7e:6720
706b:dc6:dc20:1727:99a6:db7f:79ca:3c19
d48d:1cf8:7fd3:527b:e81f:1cd5:ddd4:ac69
4a1e:70fe:9494:a0b5:4c88:6e8:7c63:ac10
```

```
2bf3:dd0f:1810:a913::4002:1
14a9:d441:bba:2856::4000:8080:41
d273:f263:8440:fa2c:68:200:4000:101
b738:4db6:a21:a007::1
62ba:2e95:ed41:cfd5:8::401:a1
```

Next Steps:

- Update your solution
- Do not copy-paste this sample solution