

Python

call by object reference

- Call by value
- Call by reference
- Call by object reference

왜 'object' reference 일까?

What is Object?

Python 에서는 다음과 같이 정의하고 있다.

Any data with state (attributes or value) and defined behavior (methods).
Also the ultimate base class of any new-style class

특징 또는 값을 가진 데이터로서, 정해진 행동 양식을 가지며 새로운
클래스의 가장 기초가 되는 클래스 (~~발번역~~)

출처: <https://docs.python.org/3/glossary.html>

(파이썬 용어사전, 위의 attribute나 method 등의 용어도 정의하고 있으니 찾아보세요)

파이썬에서 객체에는 수많은 종류가...

- 기본 객체
- 숫자 객체
- 시퀀스 객체
- 컨테이너 객체
- 함수 객체
- 기타 객체
-

~~이 또한 파이썬의 위엄이겠지요 ○ 스피너~~



<https://docs.python.org/3/c-api/concrete.html>

기본 객체만 살펴보자

- **형 객체 (Type object)**
 - PyObject* PyType_Type (형 객체의 객체)
 - int PyType_Check ...
 - 너무 많음
- **None 객체 (None object)**
 - None도 객체다!
 - PyObject* Py_None
 - Py_RETURN_NONE



<https://docs.python.org/ko/3/c-api/type.html>

PyObject의 구성

```
typedef struct _object {  
    _PyObject_HEAD_EXTRA  
    Py_ssize_t ob_refcnt;  
    PyTypeObject *ob_type;  
} PyObject;
```

```
typedef struct {  
    PyObject ob_base;  
    Py_ssize_t ob_size; /* Number of items in variable part */  
} PyVarObject;  
  
/* Cast argument to PyVarObject* type. */  
#define _PyVarObject_CAST(op) ((PyVarObject*)(op))  
#define _PyVarObject_CAST_CONST(op) ((const PyVarObject*)(op))
```

<https://github.com/python/cpython/blob/master/Include/object.h>

PEP-8 (파이썬 국룰)

부제 : 팀원을 위한 코딩



[PEP 8 -- Style Guide for Python Code | Python.org](https://www.python.org/dev/peps/pep-0008/)

Imports

```
# Correct:  
import os  
import sys
```

```
# Wrong:  
import sys, os
```

변수 지정(처음 이름 정하는 것이 중요!!)

- 소문자 L, 대문자 O, 대문자 I는 변수명으로 사용하지 마세요. 어떤 폰트에서는 가독성이 굉장히 안 좋습니다.

```
x = 'John Smith'  
y,z = x.split()  
#
```



```
name = 'John Smith'  
first_name, last_name = name.split()
```

```
x = 2 # 수학 식 사용 할 때만  
my_variable = 'two'  
  
GRAVITY_ACCEL = 9.8 #constant(상수)
```

```
random_=random.randint(10,100) #내장함수와 겹치면 그 뒤에 _  
|
```

Code lay-out

- 들여쓰기는 공백 4칸을 권장합니다.
- 한 줄은 최대 79자까지
- 최상위(top-level) 함수와 클래스 정의는 2줄씩 띄어 씁니다.
- 클래스 내의 메소드 정의는 1줄씩 띄어 씁니다.

```
def db(x): #-> multiply_by_two(x):
    return x * 2
```

```
2
3
4 def add_function(a,b):#소문자 사용, _사용
5     |
6     return a+b
7
8
```

```
32
33
34 class MyName: # 카멜케이스로 작성
35
36     def __init__(self, name):
37
38         self.name = name
39
40     def __str__(self):
41
42         return self.name
43
44
```

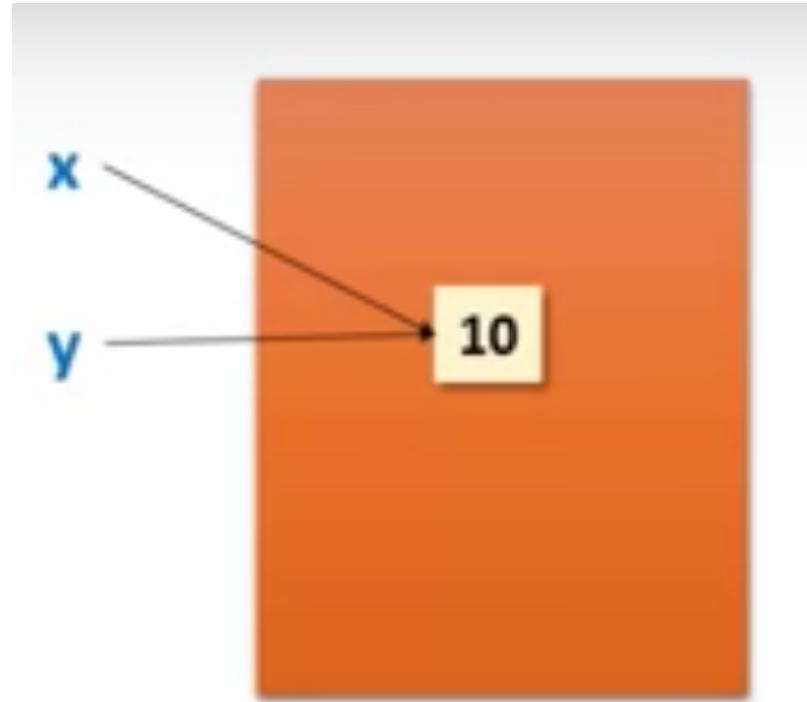
기타

Comments

- 코드와 모순되는 주석은 없느니만 못합니다. 항상 코드에 따라 갱신해야 합니다.
- 불필요한 주석은 달지 마세요.
- 한 줄 주석은 신중히 다세요.
- 다음과 같은 곳의 불필요한 공백은 피합니다.
 - 대괄호([])와 소괄호(())안
 - 쉼표(,), 쌍점(:)과 쌍반점(;) 앞

Memory management in Python

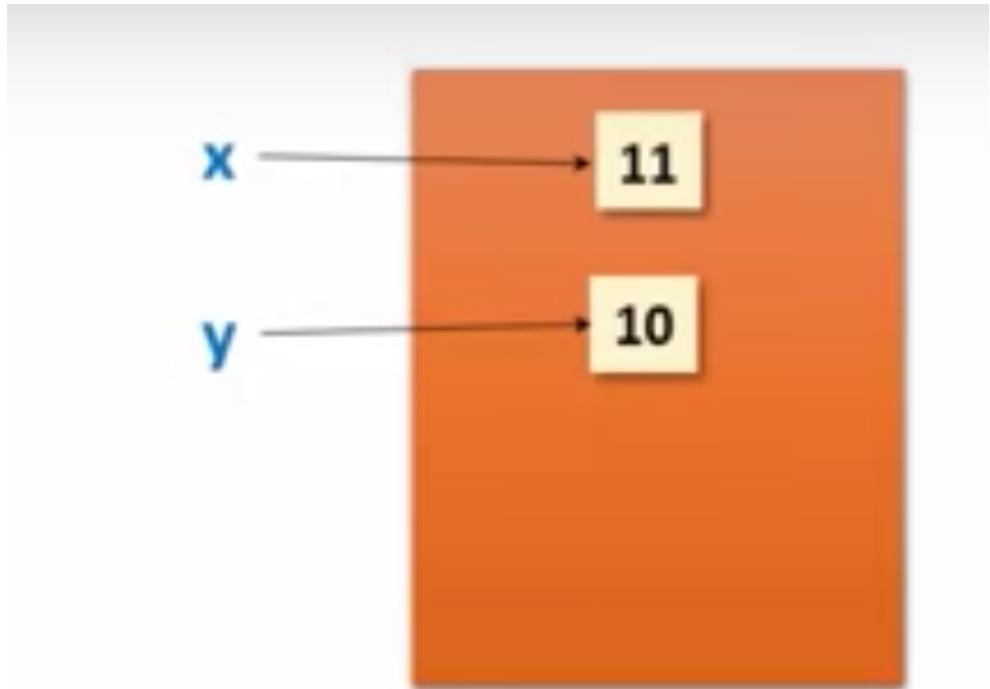
```
x=10  
  
y=x  
if id(x) == id(y):  
    print('x and y refer to the same object')
```



x and y refer to the same object

```
x=10  
  
y=x  
if id(x) == id(y):  
    print('x and y refer to the same object')  
  
x = x + 1  
if id(x) != id(y):  
    print(['x and y refer to DIFFERENT object'])
```

x and y refer to the same object
x and y refer to DIFFERENT object

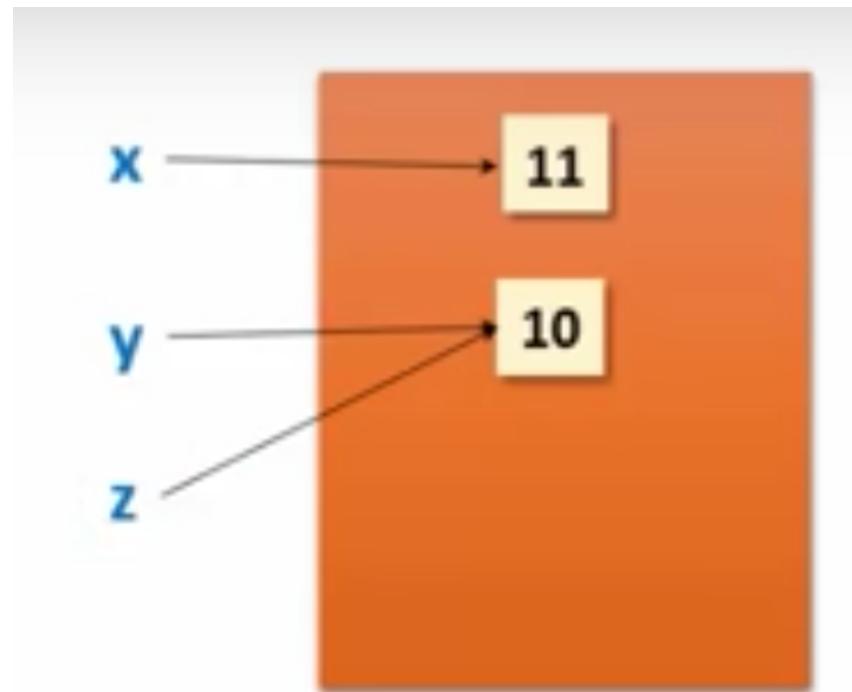


```
x=10

y=x
if id(x) == id(y):
    print('x and y refer to the same object')

x = x + 1
if id(x) != id(y):
    print('x and y refer to DIFFERENT object')

z = 10
if id(y) == id(z):
    print('y and z refer to the same object')
else:
    print(['y and z refer to DIFFERENT object'])
```



```
x and y refer to the same object
x and y refer to DIFFERENT object
y and z refer to the same object
```

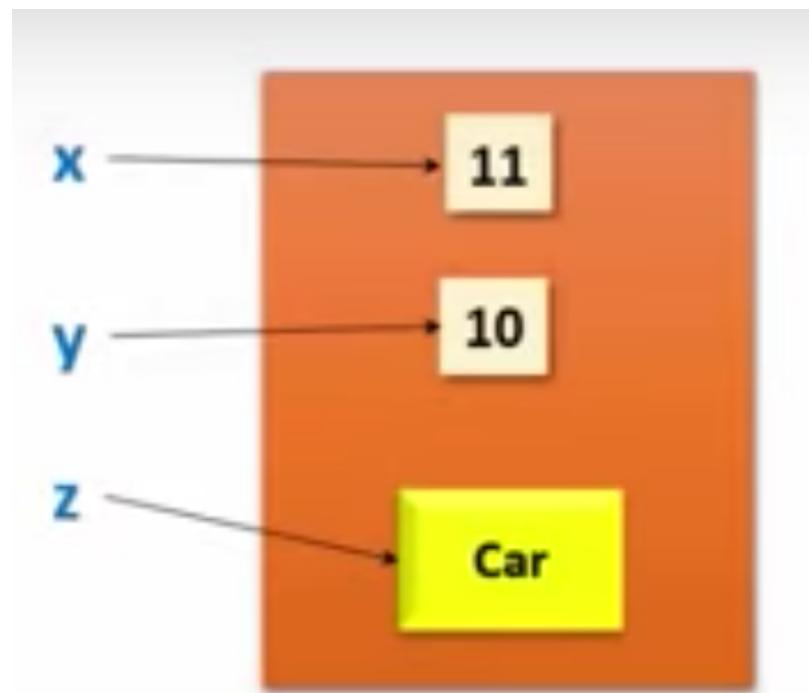
```
x=10

y=x
✓ if id(x) == id(y):
|   print('x and y refer to the same object')

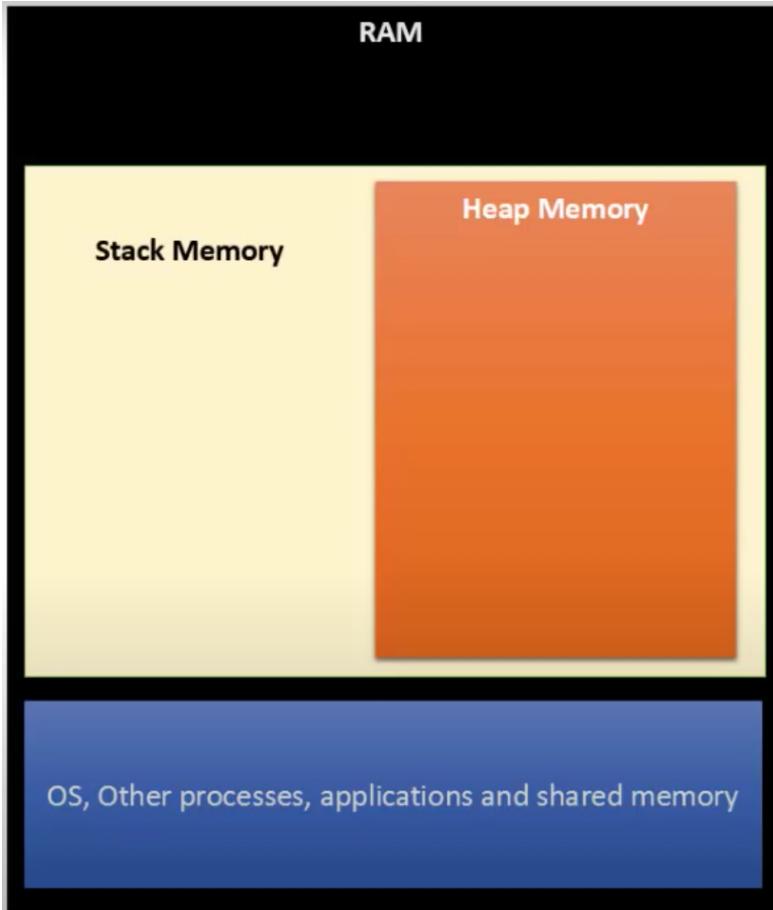
x = x + 1
✓ if id(x) != id(y):
|   print('x and y refer to DIFFERENT object')

z = 10
✓ if id(y) == id(z):
|   print('y and z refer to the same object')
✓ else:
|   print('y and z refer to DIFFERENT object')

z = Car() # 차 클래스가 있다면
```



S



Stack memory: methods, variables

Heap memory: objects, instance variable

Stack frame: method, function 호출 시 생김

- Stack frame은 method/function return 시 사라짐

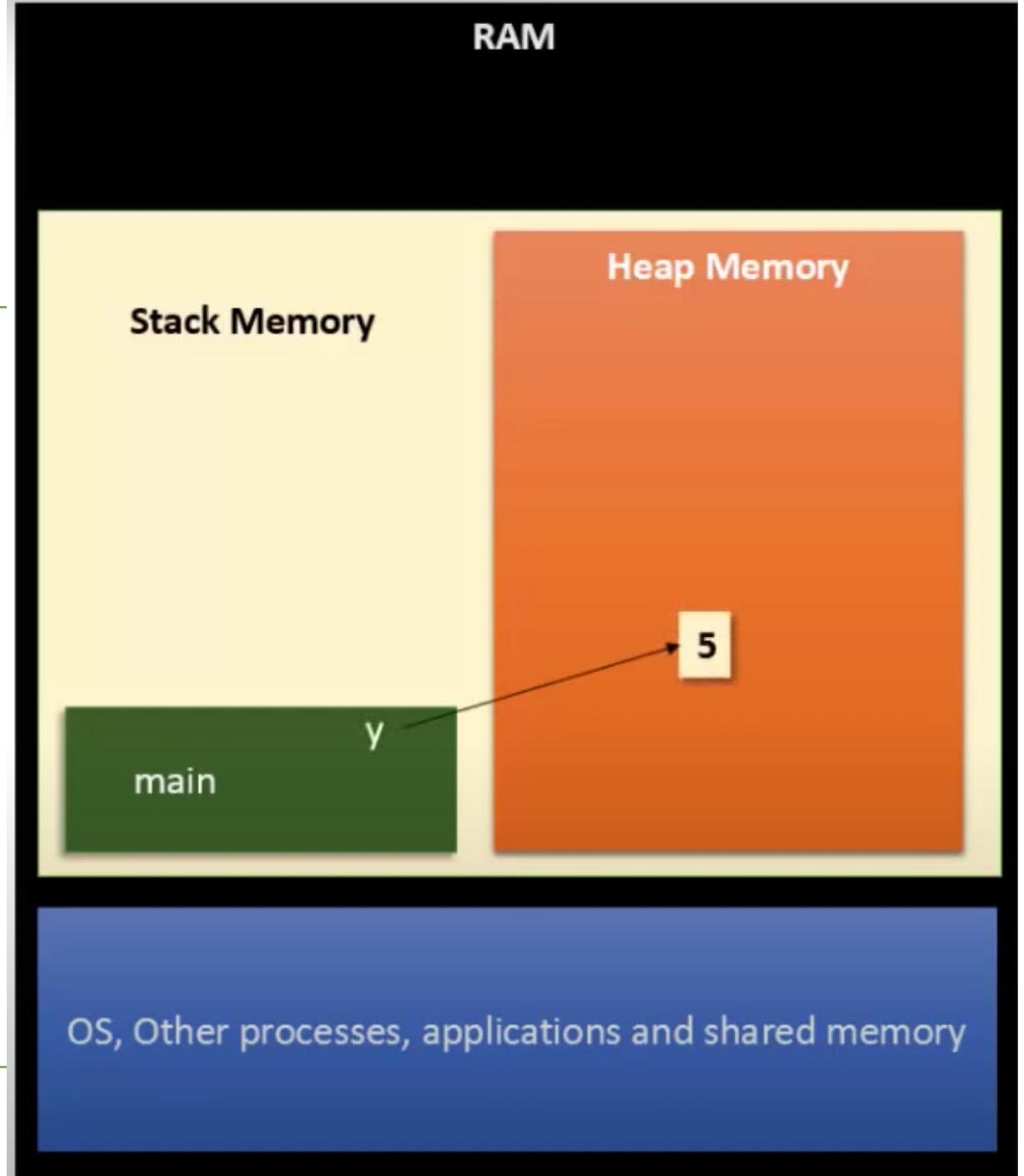
- # main
- y = 5

Stack memory: methods, variables

Heap memory: objects, instance variable

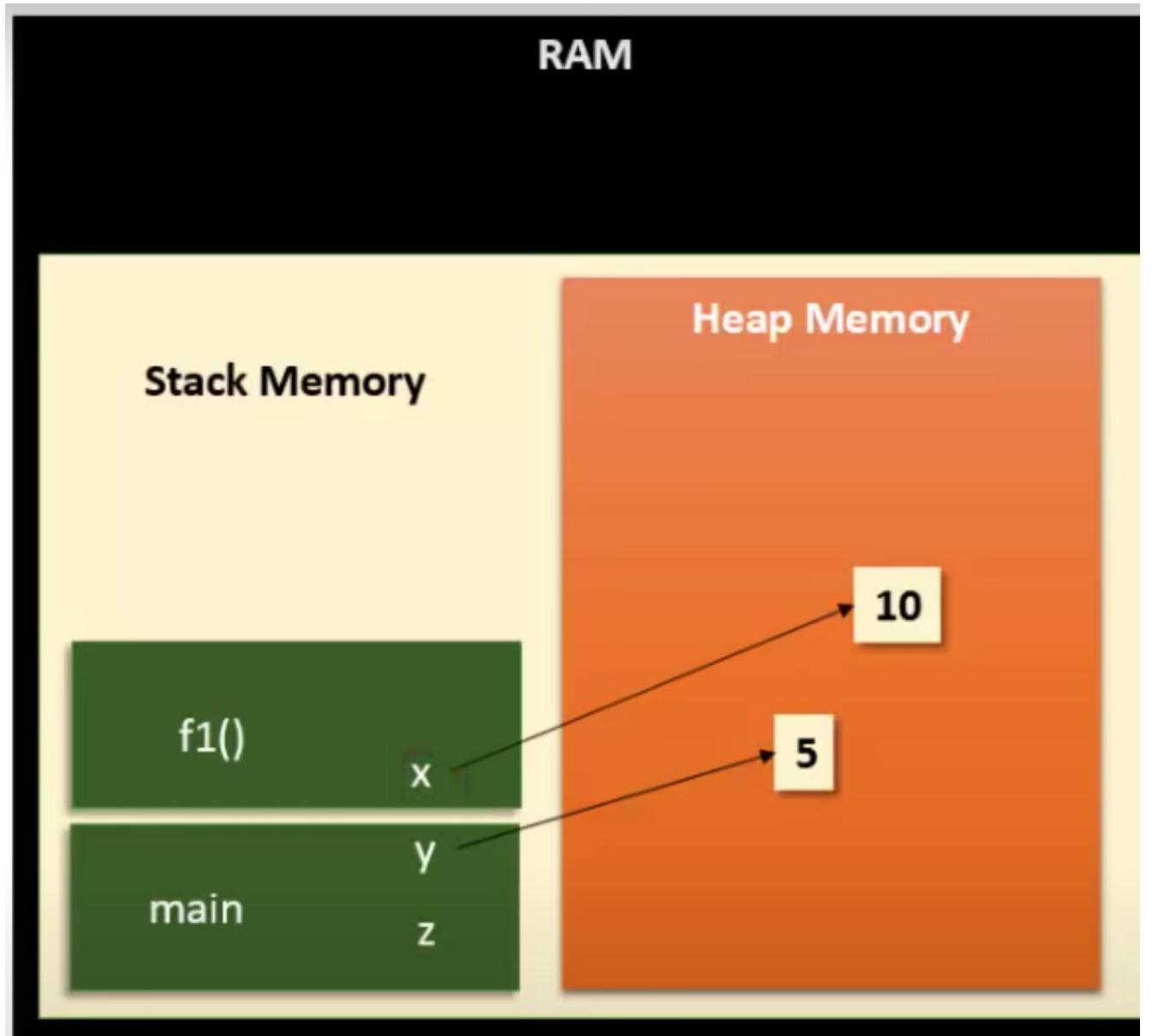
Stack frame: method, function 호출 시 생김

- Stack frame은 method/function return 시 사라짐



```
def f1(x):
    x = x*2
    y = f2(x)
    return y

# main
y = 5
z = f1(y)
```



Stack memory: methods, variables

Heap memory: objects, instance variable

Stack frame: method, function 호출 시 생김

- Stack frame은 method/function return 시 사라짐

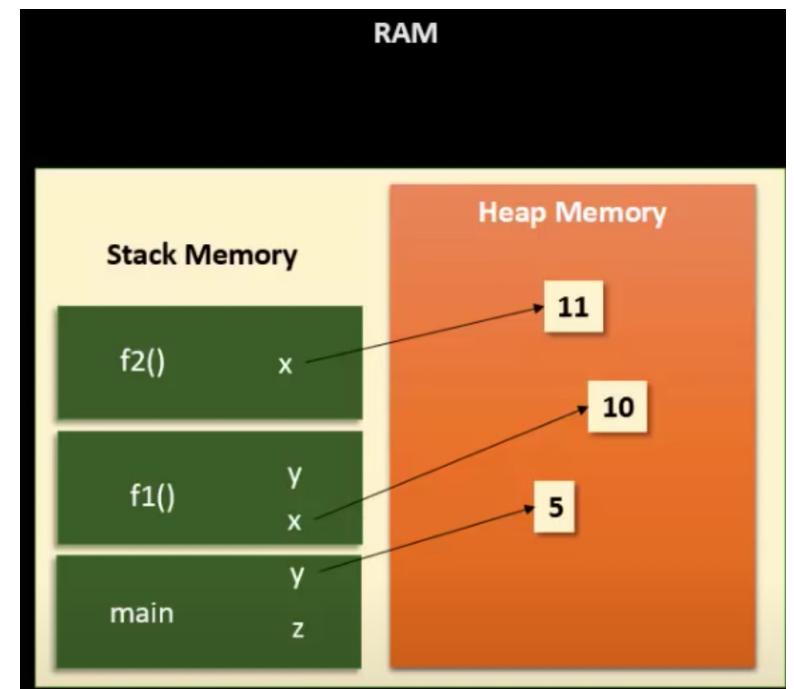
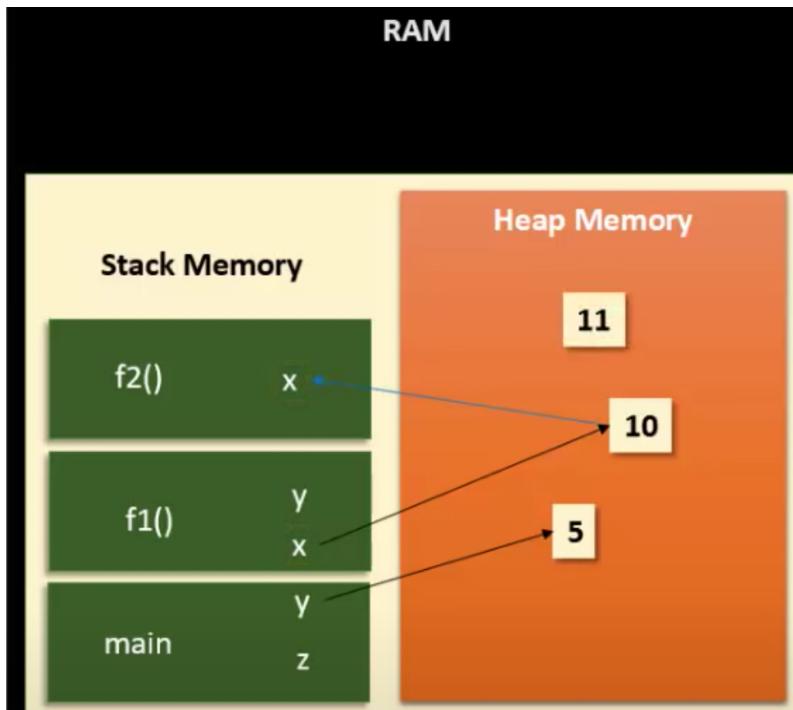
```

def f2(x):
    x = x + 1
    return x

def f1(x):
    x = x*2
    y = f2(x)
    return y

# main
y = 5
z = f1(y)

```



Stack memory: methods, variables

Heap memory: objects, instance variable

Stack frame: method, function 호출 시 생김

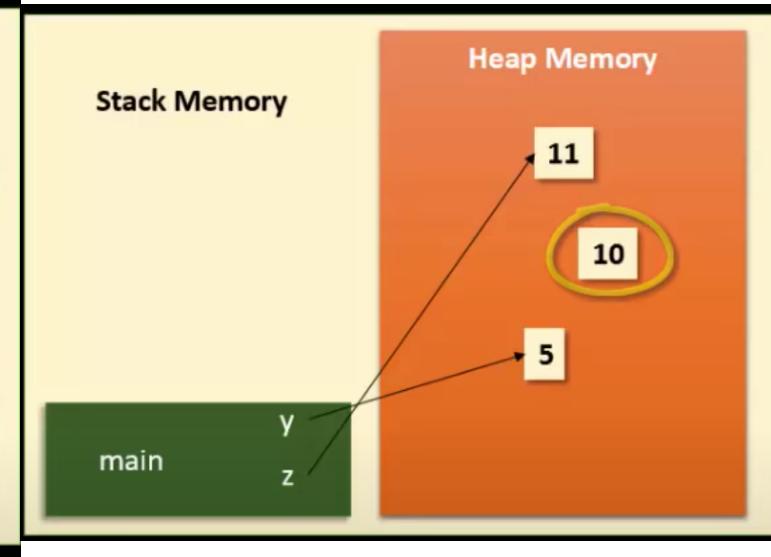
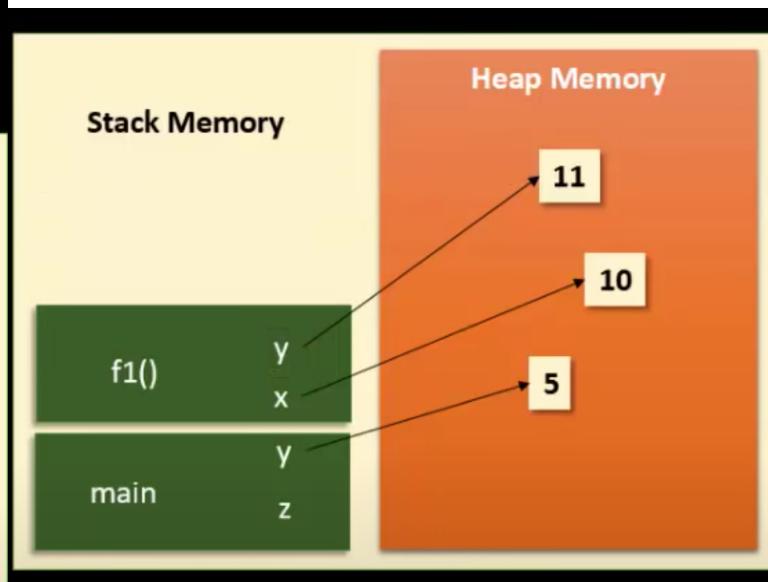
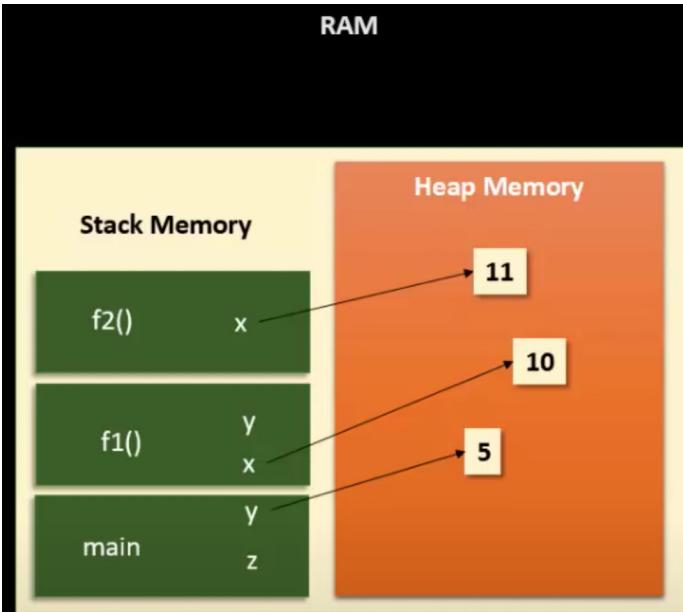
- Stack frame은 method/function return 시 사라짐

함수 호출이 끝나면

```
def f2(x):
    x = x + 1
    return x

def f1(x):
    x = x*2
    y = f2(x)
    return y

# main
y = 5
z = f1(y)
```



Stack memory: methods, variables

Heap memory: objects, instance variable

Stack frame: method, function 호출 시 생김

- Stack frame은 method/function return 시 사라짐

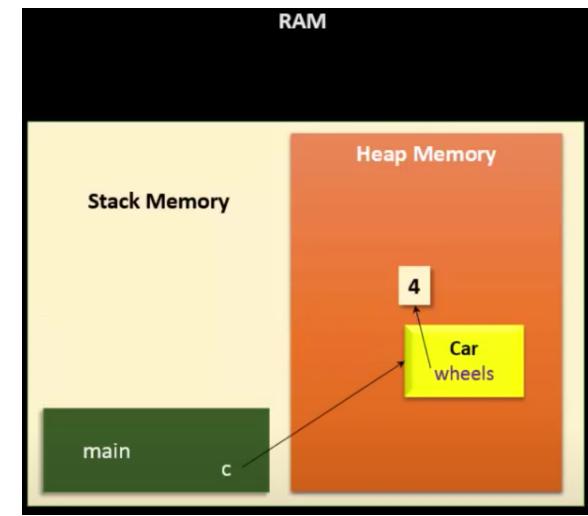
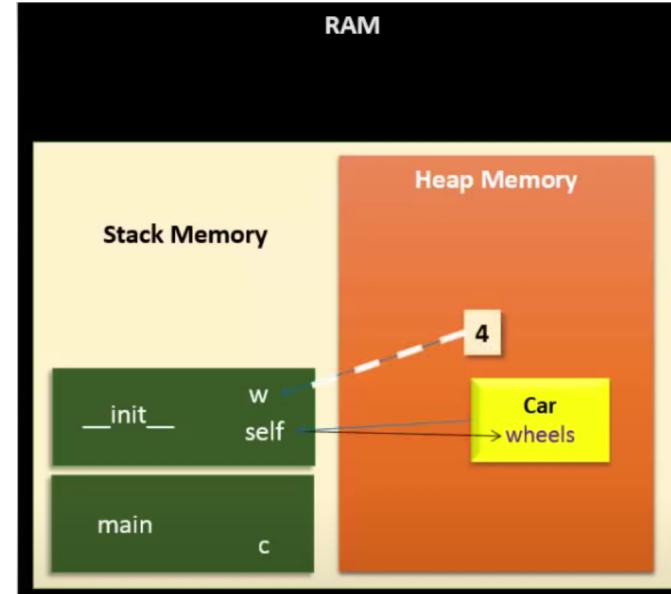
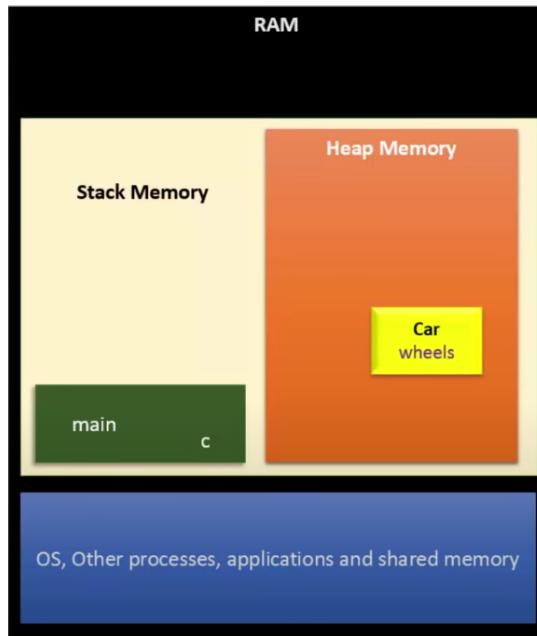
여기서 object 10의 운명은?

```

class Car:
    def __init__(self,w):
        self.wheels = w
    def getWheels(self):
        return self.wheels

#main
c = Car(4)

```



Stack memory: methods, variables

Heap memory: objects, instance variable

Stack frame: method, function 호출 시 생김

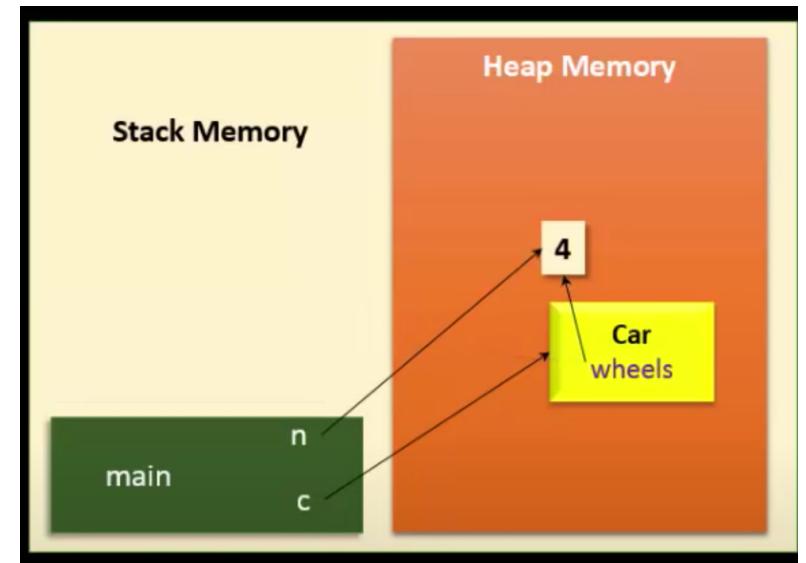
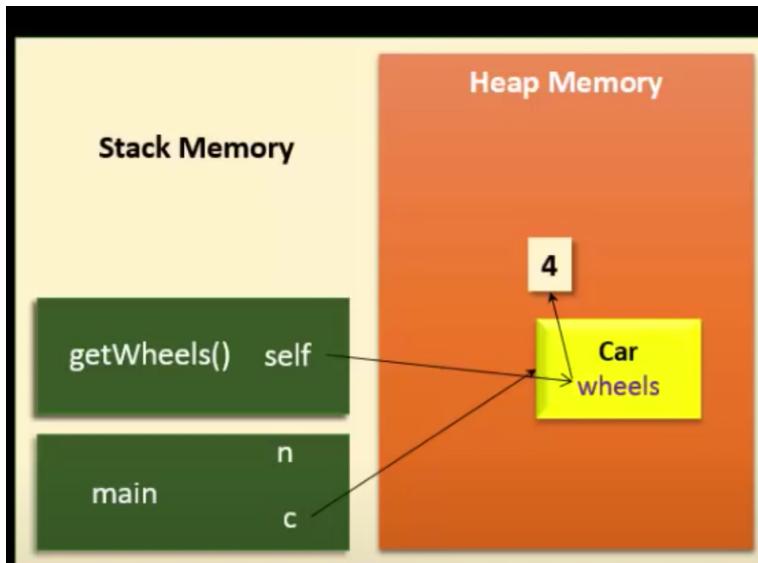
- Stack frame은 method/function return 시 사라짐

```

class Car:
    def __init__(self,w):
        self.wheels = w
    def getWheels(self):
        return self.wheels

#main
c = Car(4)
n = c.getWheels()

```



Stack memory: methods, variables

Heap memory: objects, instance variable

Stack frame: method, function 호출 시 생김

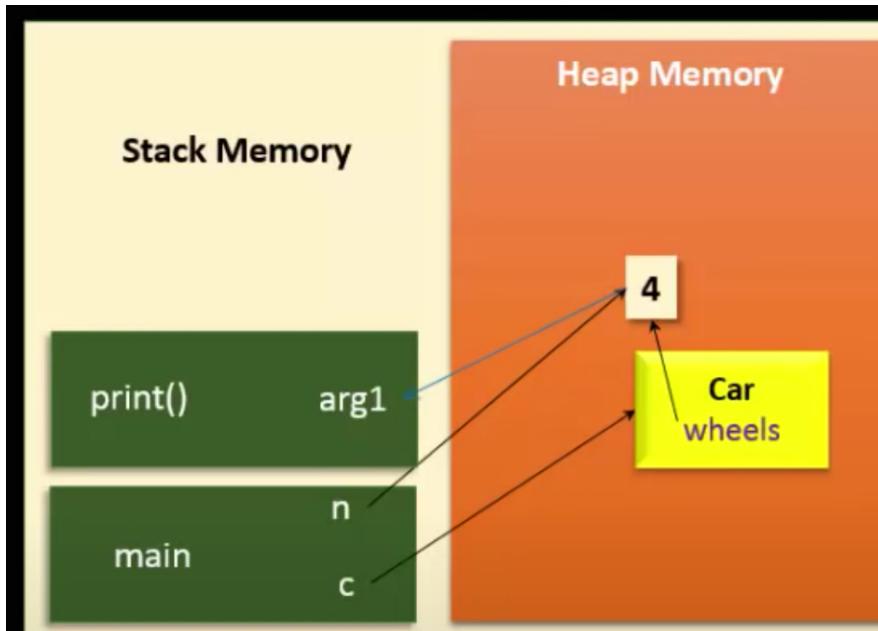
- Stack frame은 method/function return 시 사라짐

```

class Car:
    def __init__(self,w):
        self.wheels = w
    def getWheels(self):
        return self.wheels

#main
c = Car(4)
n = c.getWheels()
print(n)

```



Stack memory: methods, variables

Heap memory: objects, instance variable

Stack frame: method, function 호출 시 생김

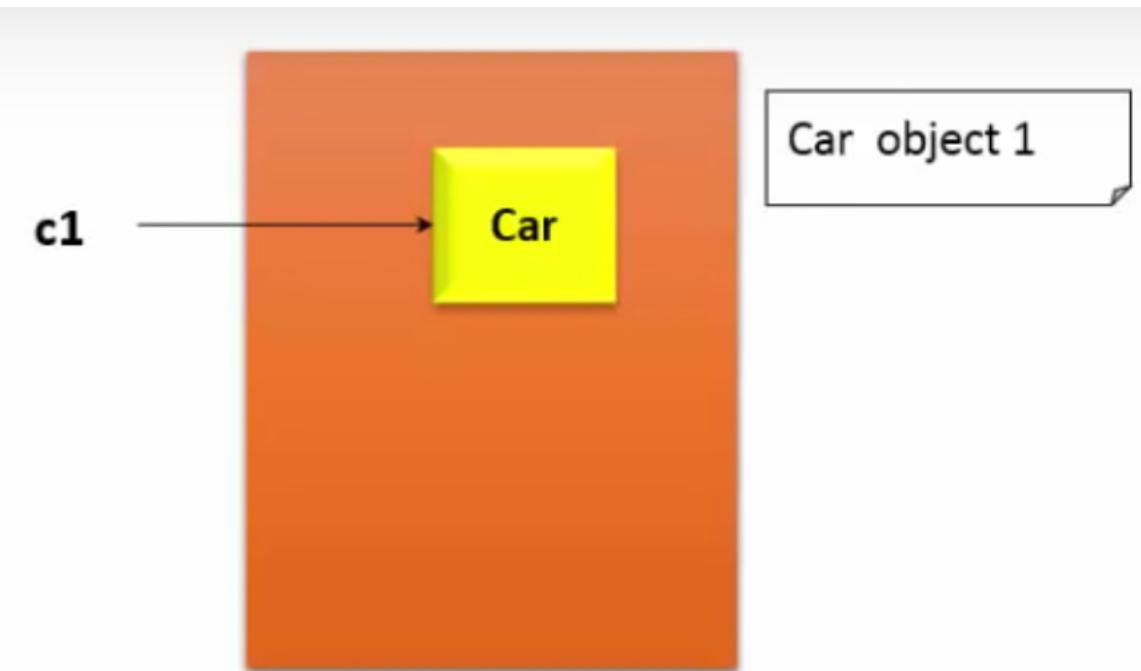
- Stack frame은 method/function return 시 사라짐

Garbage collector

```
class Car:  
    def __init__(self,w):  
        self.wheels = w  
  
    def getWheels(self):  
        return self.wheels
```

```
c1 = Car(4)
```

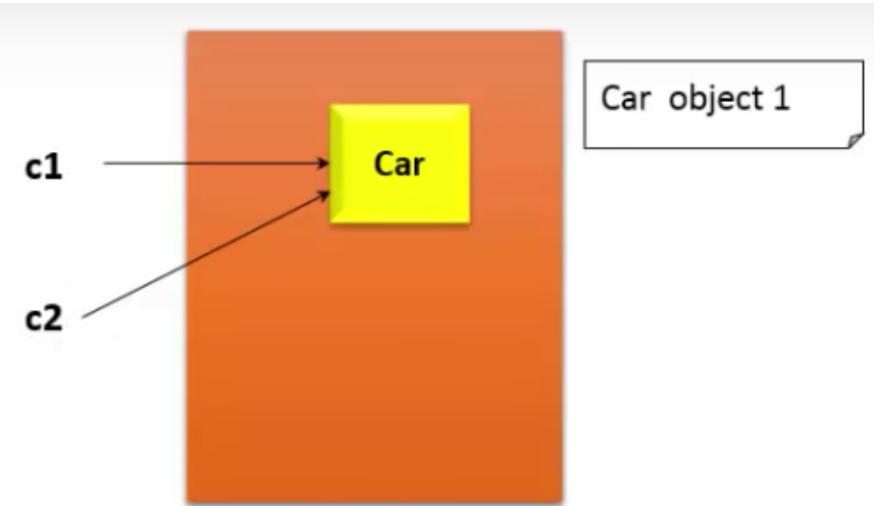
Object	Number of references
Car object 1	1



Garbage collector

```
class Car:  
    def __init__(self,w):  
        self.wheels = w  
  
    def getWheels(self):  
        return self.wheels
```

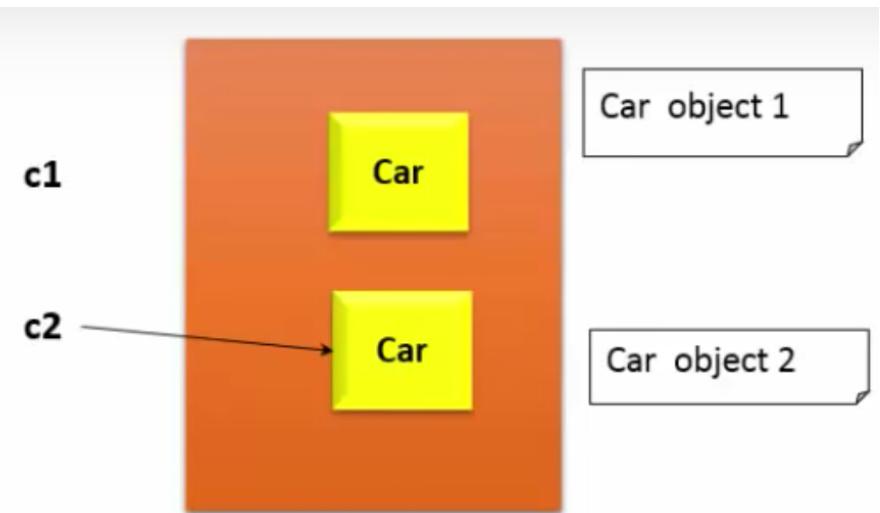
```
c1 = Car(4)  
c2 = c1
```



Object	Number of references
Car object 1	1 + 1

```
class Car:  
    def __init__(self,w):  
        self.wheels = w  
  
    def getWheels(self):  
        return self.wheels
```

```
c1 = Car(4)  
c2 = c1  
c1 = None  
c2 = Car(4)
```

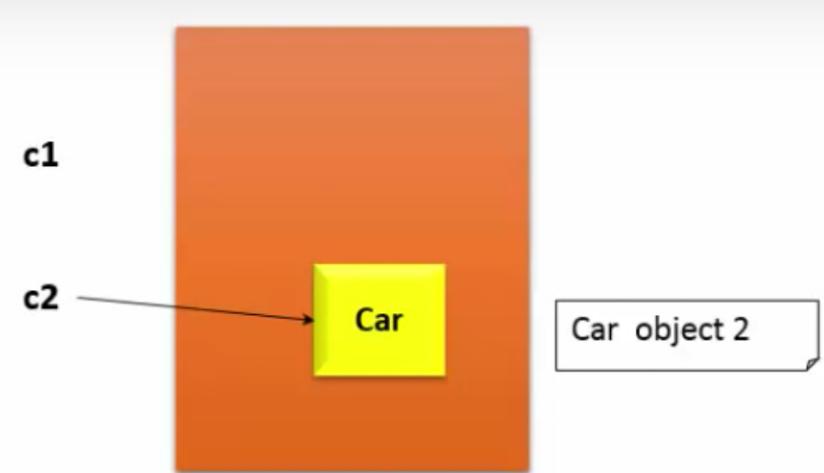


Object	Number of references	GC
Car object 1	0	Dead Object
Car object 2	1	



```
class Car:  
    def __init__(self,w):  
        self.wheels = w  
  
    def getWheels(self):  
        return self.wheels
```

```
c1 = Car(4)  
c2 = c1  
c1 = None  
c2 = Car(4)
```



Object	Number of references	GC
Car object 1	0	
Car object 2	1	

The algorithm used for
Garbage Collection (GC)
is called
Reference Counting



Further study

- Stack memory vs heap memory
- Garbage collector
- Memory leak
- Stackoverflow
- Pointer

reference

- https://www.youtube.com/watch?v=arxWaw-E8QQ&feature=youtu.be&ab_channel=simplefunde

Double Underscore

Single underscore vs Double underscore

Special method

Module name

Single underscore VS Double underscore

	Single(_)	Double(_)
Meaning	Special variables and methods	
Leading using	<ul style="list-style-type: none">- this is PRIVATE with convention defined in PEP8(Naming Convention에 대한 규약)- Why convention? PYTHON은 java처럼 public과 private에 대한 철저한 분리가 존재하지 않는다. <ul style="list-style-type: none">- import 또는 class usage: 구체적인 namespace, module scope가 주어져야 사용 가능한 PRIVATE를 흉내내는 special method (다음 슬라이드 참고) → JAVA의 PRIVATE는 접근 자체가 안됨.	<ul style="list-style-type: none">- Mangling 목적으로 사용된다. 싱글 언더스코어와 비슷한데, 이 경우에는 특수한 처리 없이 접근 자체가 안된다.→ 이와 같은 Mangling은 _classname_variable(또는 method)name으로만 접근 가능하다.
Trailing using	Trailing using: Keyword(예약어)의 name과 충돌을 막기 위한 convention	- 사용 x
Leading+Trailing	- 사용 x	- Magic method(=built-in method, ref 참고!)

Avoid Wildcard '*'

- tempModule.py
- mainFlow.py

```
mainFlow.py
1  from tempModule import *
2
3  print("Here is the mainflow")
4  run()
5
6  external_func()
> 7  _internal_func()
```

예외가 발생했습니다. **NameError**

name '_internal_func' is not defined

File "C:\Users\주찬형\peersession\day2\mainFlow.py", line 7, in <module>
 _internal_func()

```
def run():
    print("tempModule is running")

def external_func():
    print("external")
def _internal_func():
    print("internal")
if __name__ == "__main__":
    print("here is the tempModule")
else:
    print("tempModule is imported")
```

```
def list_(x):
    return n
#used by convention to avoid conflicts with Python keyword
```

list나 class등 python의 예약어를 함수명으로 사용하고 싶을 때 뒤에 언더스코어 1개를 붙인다.

```
1  class MyClass():
2      def __init__(self):
3          self.__superprivate="Hello"
4          self._semiprivate=", world!"
5
```

```
1  from MyClass import *
2  mc=MyClass()
3  print(mc._semiprivate)
4  #print(mc.__superprivate)
5  print(mc._MyClass__superprivate)
```

→ 주석처리한건 attribute error

→ 맹글링 처리한 건 정상 접근 가능. (single leading underscore과 다르게
직접 인스턴스 생성해서 스코프 만들어줘도 접근 자체가 안된다.)

Module Name : `__name__`

The screenshot shows a code editor interface with two files open:

- `mainFlow.py`:

```
1 import tempModule as tm
2
3 print("Here is the mainflow")
4 tm.run()
```
- `tempModule.py`:

```
1 def run():
2     print("tempModule is running")
3
4 if __name__ == "__main__":
5     print("here is the tempModule")
6 else:
7     print("tempModule is imported")
```

The terminal window displays the output of running the `mainFlow` script:

```
FLOW.py
tempModule is imported
Here is the mainflow
tempModule is running
PS C:\Users\조주현\OneDrive\
```

The screenshot shows the `tempModule.py` file from the previous editor:

```
1 def run():
2     print("tempModule is running")
3
4 if __name__ == "__main__":
5     print("here is the tempModule")
6 else:
7     print("tempModule is imported")
```

The terminal window displays the output of running the `tempModule` script directly:

```
tempModule.py
here is the tempModule
PS C:\Users\조주현\OneDrive\
```

Reference

- <https://doorbw.tistory.com/153> , 언더스코어란?
- <https://dojang.io/mod/page/view.php?id=2448> , 코딩도장: 모듈과 시작점 알아보기
- <https://dbader.org/blog/meaning-of-underscores-in-python> , The meaning of underscore in Python

Git

Git 사용이유

다른 사람과 함께 특정한 프로젝트를 동시에 작업할 수 있음

동일한 소스코드를 2명 이상이 고친다고 하더라도 효과적으로 소스 코드를 서로 합침으로써 충돌이 발생하지 않도록 함 -> 분산형 협업 도구

-장점

Branch 덕분에 프로젝트의 가지치기가 가능

작업된 모든 변경 내역이 저장되어 안전하게 프로젝트를 운영할 수 있음

오픈소스

오픈 소스와 오픈소스 라이선스

<https://www.youtube.com/watch?v=bm2a5fozcms>

오픈 소스를 사용하면 어쩔 수 없이 소스코드를 공개해야 하는 경우도 있음

Git 저장소

프로젝트 파일들을 올리고 그 파일들을 관리하는 저장소

일반적으로 Git hub가 사용됨

Github 오픈 소스 프로젝트에서 많이 사용됨

Git 동작원리

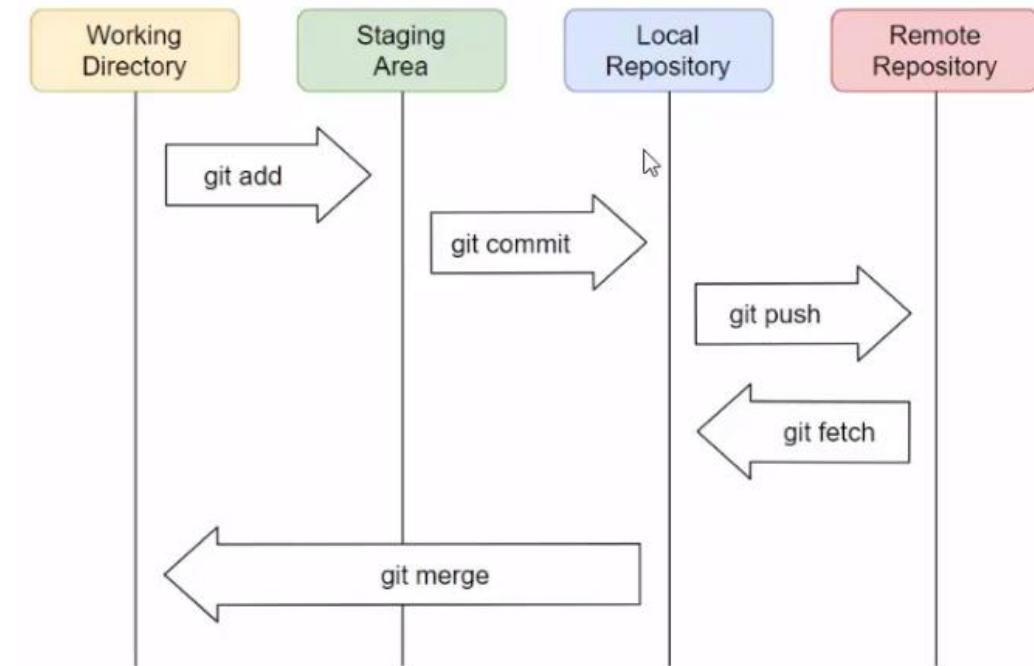
Working directory

: 작업할 파일이 있는 디렉토리

Staging area

: commit을 수행할 파일들이 올라가는 영역

Add했을 때 add한 애들이 있는 곳



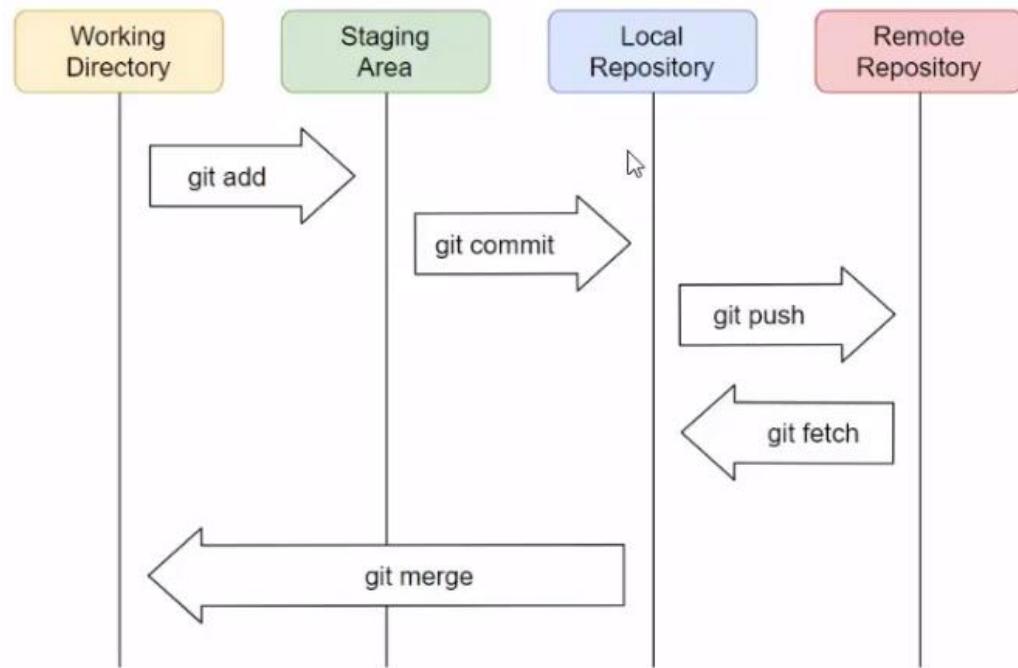
Git directory

: .git폴더, git 프로젝트의 메타 데이터와 데이터 정보(커밋의 해시 등)가 저장되는 디렉토리

Git 동작원리

동시에 파일 수정을 진행해서
Local repository에 있는 파일들과
Fetch한 파일들이 다를 때 merge

Git fetch + merge 한번에 = pull



Git 설치

Git을 사용하기 위해서는 먼저 git을 설치해주어야 함
Git 홈페이지에서 다운로드 및 설치가 가능

Setup 창의 각 옵션들에 대해 잘 설명해 준 블로그
<https://goddaehee.tistory.com/216>

Git 버전 확인

```
git --version
```

```
(boostcamp) C:\Users\rlfgm>git --version  
git version 2.30.0.windows.2
```

Git 환경설정 Git Config

git config --list

현재 git의 환경설정 확인

git config --global 환경설정

해당 컴퓨터의 전체에 환경설정

```
Anaconda Prompt (anaconda3)
(bootcamp) D:\#기리\workspace\test\Git-Tutorial>git config --list
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=false
pull.rebase=false
credential.helper=manager-core
credential.https://dev.azure.com/usehttppath=true
init.defaultbranch=master
user.name=hee
user.email=r1fgm1tkfkd@naver.com
core.repositoryformatversion=0
core.filemode=false
core.bare=false
core.logallrefupdates=true
core.symlinks=false
core.ignorecase=true
remote.origin.uri=https://github.com/grazerhee/Git-Tutorial.git
remote.origin.fetch=+refs/heads/*:refs/remotes/origin/*
branch.master.remote=origin
branch.master.merge=refs/heads/master

(bootcamp) D:\#기리\workspace\test\Git-Tutorial>
```

Git 환경설정

git config --global user.name hee

유저 이름 설정

git config --global user.email 이메일

유저 메일 설정

환경설정을 해주면 안정적으로 깃을 사용할 수 있는 상태가 됨

Git 환경설정

이러한 config 환경설정은
Config 파일로 저장되어 있음

global config 파일은
사용자의 홈 디렉토리 내에 gitconfig라는 파일로써 존재한다.

각 repository에 따른 환경설정도 .git 폴더 내의 config 파일에 저장되어 있다.

Git 환경설정 Git Config

동일한 환경설정에 대해

Global과 repository 내의 값이 다르면

Git config --list에는 둘 다 나오지만

실제로는 프로젝트 내의 환경설정을 따른다.



The screenshot shows a terminal window titled "Anaconda Prompt (anaconda3)". The command `git config --global --list` is run, displaying global configuration settings:

```
(boostcamp) D:\#기리\workspace\test\Git-Tutorial>git config --global --list
user.name=hee
user.email=r1fgm1tkfkfd@naver.com
```

Then, the command `git config user.name=gilhee` is run, resulting in an error:

```
(boostcamp) D:\#기리\workspace\test\Git-Tutorial>git config user.name=gilhee
error: invalid key: user.name=gilhee
```

Finally, the command `git config user.name "gilhee"` is run successfully:

```
(boostcamp) D:\#기리\workspace\test\Git-Tutorial>git config user.name "gilhee"
```

Git 환경설정 Git Config

```
■ Anaconda Prompt (anaconda3)
(boilstcamp) D:\BoostCamp\workspace\test\Git-Tutorial>git config --list
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=false
pull.rebase=false
credential.helper=manager-core
credential.https://dev.azure.com/usehttppath=true
init.defaultbranch=master
user.name=hee
user.email=r1tgm1tkfkfd@naver.com
core.repositoryformatversion=0
core.filemode=false
core.bare=false
core.logallrefupdates=true
core.symlinks=false
core.ignorecase=true
remote.origin.url=https://github.com/grazerhee/Git-Tutorial.git
remote.origin.fetch=+refs/heads/*:refs/remotes/origin/*
branch.master.remote=origin
branch master merge=refs/heads/master
user.name=gilhee
```

Repository

repository는 실제 소스코드가 담겨 있으면서 commit 내역 등의 모든 작업 이력이 담겨 있는 공간을 의미

소스 코드가 저장되는 공간을 의미

하나의 프로젝트가 담기는 공간

Repository

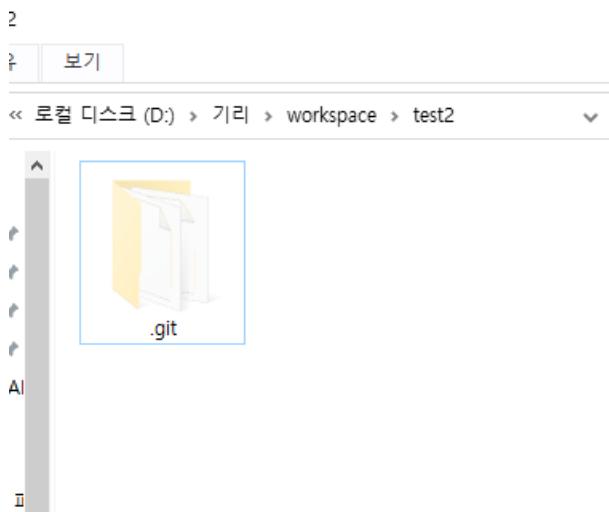
생성 방법

- 1) local 상에서 repository를 생성 후 local repository를 remote repository와 연결
- 2) Remote repository에서 생성 후 local로 clone

Repository

생성 방법

- 1) local 상에서 repository를 생성 후 local repository를 remote repository와 연결
repository로 만들고 싶은 directory 내에서
git init 실행



```
Anaconda Prompt (anaconda3)
(bootcamp) D:\기리\workspace\test2>git status
fatal: not a git repository (or any of the parent directories): .git

(bootcamp) D:\기리\workspace\test2>git init
Initialized empty Git repository in D:/기리/workspace/test2/.git/

(bootcamp) D:\기리\workspace\test2>git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)

(bootcamp) D:\기리\workspace\test2>
```

Repository

생성 방법

- 1) local 상에서 repository를 생성 후 local repository를 remote repository와 연결
아직 local repository만 존재하므로 remote repository와 연결해 주어야한다
git remote add origin <https://github.com/>본인계정/프로젝트이름



The screenshot shows a terminal window titled "Anaconda Prompt (anaconda3)". The command entered is "git remote add origin https://github.com/grazherhee/Git-Tutorial". The command has been partially typed, with the URL and repository name visible.

```
(boostcamp) D:\기리\workspace\test2>git remote add origin https://github.com/grazherhee/Git-Tutorial
(	boostcamp) D:\기리\workspace\test2>
```

Repository

생성 방법

2) Remote repository에서 생성 후 local로 clone

Repository

Owner * Repository name *

 grazerhee ✓ / Git-Tutorial ✓

Great repository names are short and memorable. Need [inspiration](#)? How about [expert-spork](#)?

Description (optional)

 Public
Anyone on the internet can see this repository. You choose who can commit.

 Private
You choose who can see and commit to this repository.

Repository의 이름을 설정

Repository에 대한 설명을 넣을 수 있음

Public/private

Public을 해야 누구나 이 repository를 볼 수 있는 open-source가 됨

Repository

Initialize this repository with:

Skip this step if you're importing an existing repository.

Add a README file

This is where you can write a long description for your project. [Learn more.](#)

Add .gitignore

Choose which files not to track from a list of templates. [Learn more.](#)

Choose a license

A license tells others what they can and can't do with your code. [Learn more.](#)

올리고 싶지 않은 형식의 파일을 설정할 때 사용

-> 개인 정보가 들어가 있는 파일이나 동영상 같이 용량이 너무 큰 것들

오픈 소스 라이선스 중 어떤 걸 선택할지 선택

Repository

Quick setup — if you've done this kind of thing before

 Set up in Desktop

or

HTTPS

SSH

<https://github.com/grazerhee/Git-Tutorial.git>

웹 주소 형태로 접근 가능

Ssh 형태로 접근 가능

Git clone <https://github.com/본인계정/>프로젝트이름
로 remote repository를 local에 복사

```
(boostcamp) D:\#기리\workspace>git clone https://github.com/grazerhee/Git-Tutorial
Cloning into 'Git-Tutorial'...
remote: Enumerating objects: 21, done.
remote: Counting objects: 100% (21/21), done.
remote: Compressing objects: 100% (15/15), done.
remote: Total 21 (delta 4), reused 20 (delta 3), pack-reused 0
Receiving objects: 100% (21/21), done.
Resolving deltas: 100% (4/4), done.
(	boostcamp) D:\#기리\workspace>
```

Remote repository 수정

1) 해당 프로젝트에 소속된 사람이 아닌 경우

commit으로 repo에 적용할 권한이 없음

소스코드를 수정하는 것에 제약 있음

이런 경우 PR(Pull Request)를 작성하여 오픈소스에 기여 가능

PR에 수정 사항 등을 담아서 전송하면

해당 오픈소스의 관리자가 이를 허용했을 때 실제로 오픈소스에 반영이 될 수 있음

PR 방법

<https://velog.io/@zansol/Pull-Request-%EC%9D%B4%ED%95%B4%ED%95%98%EA%B8%B0>

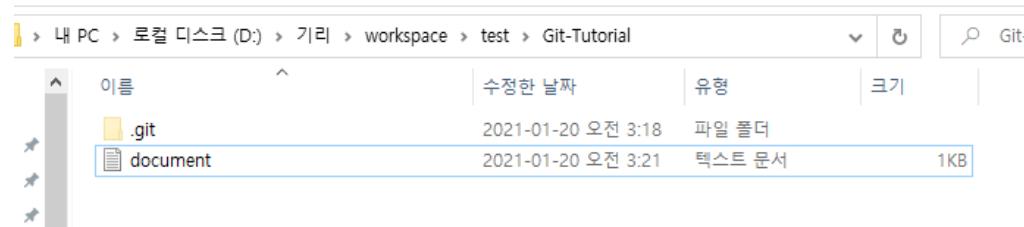
Remote repository 수정

2) 해당 프로젝트에 소속된 경우

commit -> push

Git 변경사항 적용

예를 들어 다음과 같이 local repository에 새로운 파일을 추가하면



git status 창에 변경사항이 있다고 뜸

```
(boostcamp) D:\기리\workspace\test\Git-Tutorial>git status
On branch master
Your branch is up to date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    my_module.py

nothing added to commit but untracked files present (use "git add" to track)

(	boostcamp) D:\기리\workspace\test\Git-Tutorial>
```

Git 변경사항 적용

git add 파일

변경사항을 add할 수 있으며

Add한 후에 git status를 다시 실행하면
현재 staging area에 변경사항이 올라가 있음을 볼 수 있음

```
(boostcamp) D:\기리\workspace\test\Git-Tutorial>git add *.py
(bootcamp) D:\기리\workspace\test\Git-Tutorial>git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   my_module.py
```

Git 변경사항 적용

Add를 한 후 commit을 진행

Commit을 해야 실제로 변경사항이 upload되는 시점이 찍힘

git commit -m "message"

이 commit에 대한 간략한 설명을 message에 넣을 수 있음

참고로 commit을 하면 git status 창은 텅 비어있다.

```
(boostcamp) D:\기리\workspace\test\Git-Tutorial>git commit -m "Add Text File [document.txt]"
[master (root-commit) e83abdf] Add Text File [document.txt]
 1 file changed, 1 insertion(+)
 create mode 100644 document.txt

(bootcamp) D:\기리\workspace\test\Git-Tutorial>git push
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 229 bytes | 114.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/grazerhee/Git-Tutorial.git
 * [new branch]      master -> master
```

Git 변경사항 적용

그러나 아직 remote repositor에는 변경 사항이 적용되지 않음

Remote repositor에 변경 사항을 올리려면

git push를 진행

```
(boostcamp) D:\기리\workspace\test\Git-Tutorial>git commit -m "Add Text File [document.txt]"
[master (root-commit) e83abdf] Add Text File [document.txt]
 1 file changed, 1 insertion(+)
 create mode 100644 document.txt

(	boostcamp) D:\기리\workspace\test\Git-Tutorial>git push
 Enumerating objects: 3, done.
 Counting objects: 100% (3/3), done.
 Writing objects: 100% (3/3), 229 bytes | 114.00 KiB/s, done.
 Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
 To https://github.com/grazerhee/Git-Tutorial.git
 * [new branch]      master -> master
```

Add 취소

git reset 파일

명령어를 통해 add한 파일을 다시 내릴 수 있다.

즉 Unstage 됨

```
(boostcamp) D:\기리\workspace\test\Git-Tutorial>git reset my_module.py
(bootcamp) D:\기리\workspace\test\Git-Tutorial>git status
On branch master
Your branch is up to date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    my_module.py

nothing added to commit but untracked files present (use "git add" to track)
```

코드 수정 되돌리기(add 실행 전)

앞에 modified라고 뜸

```
(boostcamp) D:\기리\workspace\test\Git-Tutorial>git reset my_module.py  
  
(boostcamp) D:\기리\workspace\test\Git-Tutorial>git status  
On branch master  
Your branch is up to date with 'origin/master'.  
  
Untracked files:  
  (use "git add <file>..." to include in what will be committed)  
    my_module.py  
  
nothing added to commit but untracked files present (use "git add" to track)
```

새로운 파일 추가 시 git status

```
(boostcamp) D:\기리\workspace\test\Git-Tutorial>git status  
On branch master  
Your branch is up to date with 'origin/master'.  
  
Changes not staged for commit:  
  (use "git add <file>..." to update what will be committed)  
  (use "git restore <file>..." to discard changes in working directory)  
    modified:   my_module.py  
  
no changes added to commit (use "git add" and/or "git commit -a")
```

파일 수정 시 git status

코드 수정 되돌리기(add 실행 전)

아래의 빨간 박스에 써 있는 것처럼

Add 실행 전의 수정 파일은

Add하거나 restore해서 되돌릴 수 있음

```
(boostcamp) D:\기리\workspace\test\Git-Tutorial>git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   my_module.py

no changes added to commit (use "git add" and/or "git commit -a")
```

```
(boostcamp) D:\기리\workspace\Git-Tutorial>git restore document.txt
```

```
(boostcamp) D:\기리\workspace\Git-Tutorial>git status
On branch master
Your branch is up to date with 'origin/master'.
```

```
nothing to commit, working tree clean
```

Git log

git log를 통해 로그 기록 볼 수 있다.
엔터로 줄을 더 넘겨서 볼 수도 있고
Q로 빠져나올수 있다.

```
Anaconda Prompt (anaconda3) - git log
(boostcamp) D:\기리\workspace\test\Git-Tutorial>git log
commit 11f7759884c1889420d2c16dd3f732cc608d72f4 (HEAD -> master, origin/master)
Author: hee <r1fgmltkfkd@naver.com>
Date:   Wed Jan 20 04:58:40 2021 +0900

    Add my_module [mul]

commit c4ae004b6d9e8b3f73db8996af6d6685d69d0f1
Author: hee <r1fgmltkfkd@naver.com>
Date:   Wed Jan 20 04:54:07 2021 +0900

    Add my_module [sub]

commit d8c9ae74dec4cc5544e660ddb8ca573e1693f7e8
Author: hee <r1fgmltkfkd@naver.com>
Date:   Wed Jan 20 04:42:07 2021 +0900

    Add my_module

commit e83abdf06832a299b7e4e6a32eeb0ec785b4e453
Author: hee <r1fgmltkfkd@naver.com>
Date:   Wed Jan 20 03:25:10 2021 +0900
```

Git log 다루기

Git log에는 다양한 옵션들이 존재

Git log --stat

통계정보를 출력해준다. 해당 파일이 몇 줄이 추가 된건지

```
(boostcamp) D:\기리\workspace\test\Git-Tutorial>git log --stat
commit 29b9a75807f6ebb9cfb54adaf3c4003fa20517cc (HEAD -> master, origin/master)
Merge: d4b93df 6f2d564
Author: hee <r1fgm1tkfkd@naver.com>
Date:   Wed Jan 20 06:04:54 2021 +0900

    Add my_module [div&comment]

commit d4b93dfdae32990616b929ce35eca6f5e0a47259
Author: hee <r1fgm1tkfkd@naver.com>
Date:   Wed Jan 20 05:56:16 2021 +0900

    Add my_module [div&comment]

my_module.py | 3 ++
 1 file changed, 3 insertions(+)

commit 6f2d5649b035032200b67663e3eb2fee195305d1
Author: hee <r1fgm1tkfkd@naver.com>
```

Git log 다루기

Git log에는 다양한 옵션들이 존재
Graph 옵션은 선을 보여줌

```
Anaconda Prompt (anaconda3) - git log --graph
(boostcamp) D:\기리\workspace\test\Git-Tutorial>git log --graph
* commit 29b9a75807f6ebb9cfb54adaf3c4003fa20517cc (HEAD -> master, origin/master)
| Merge: d4b93df 6f2d564
| Author: hee <r1fgmltkfkd@naver.com>
| Date:   Wed Jan 20 06:04:54 2021 +0900
|
|       Add my_module [div&comment]
|
* commit 6f2d5649b035032200b67663e3eb2fee195305d1
| Author: hee <r1fgmltkfkd@naver.com>
| Date:   Wed Jan 20 05:48:29 2021 +0900
|
|       Add my_module [div]
|
* commit d4b93dfdae32990616b929ce35eca6f5e0a47259
| Author: hee <r1fgmltkfkd@naver.com>
| Date:   Wed Jan 20 05:56:16 2021 +0900
|
|       Add my_module [div&comment]
```

Git log 다루기

옵션 p는

commit에 적용된 항목들을 출력

뒤에 -3은 위에서부터 3개까지만 출력

```
(boostcamp) D:\기리\workspace\test\Git-Tutorial>git log -p -3
commit 29b9a75807f6ebb9cfb54adaf3c4003fa20517cc (HEAD -> master, origin/master)
Merge: d4b93df 6f2d564
Author: hee <r1fgmltkfkfd@naver.com>
Date:   Wed Jan 20 06:04:54 2021 +0900

    Add my_module [div&comment]

commit d4b93dfdae32990616b929ce35eca6f5e0a47259
Author: hee <r1fgmltkfkfd@naver.com>
Date:   Wed Jan 20 05:56:16 2021 +0900

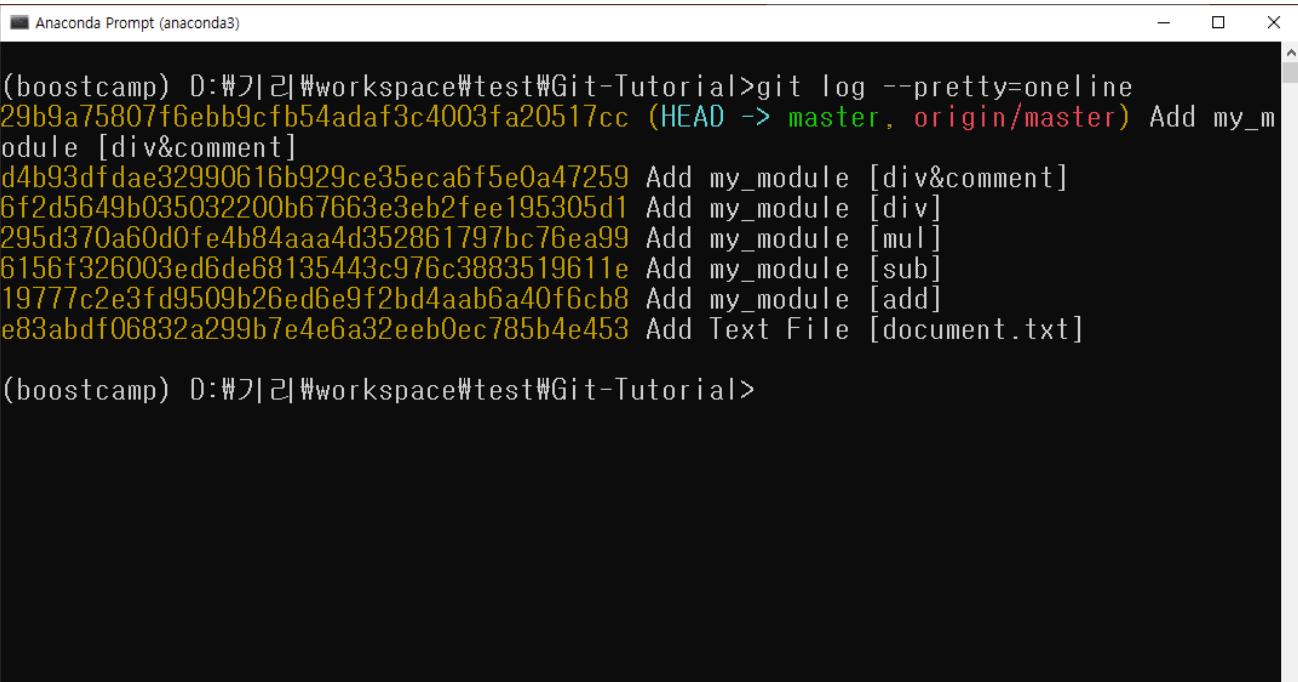
    Add my_module [div&comment]

diff --git a/my_module.py b/my_module.py
index 9b8710b..8dcb33c 100644
--- a/my_module.py
+++ b/my_module.py
@@ -6,3 +6,6 @@ def sub(a,b):
```

Git log 다루기

옵션 pretty는
원하는 대로 예쁘게 출력

이런 옵션은 따로 사이트를 참고하게 첨부하는게 좋겠다.



A screenshot of the Anaconda Prompt window titled "Anaconda Prompt (anaconda3)". The command entered is "git log --pretty=oneline". The output shows a list of commits with their hash codes, authors, dates, and commit messages. The commit messages describe the addition of files named "my_module" and "document.txt". The prompt "(boostcamp) D:\기리\workspace\test\Git-Tutorial>" is visible at the bottom.

```
(boostcamp) D:\기리\workspace\test\Git-Tutorial>git log --pretty=oneline
29b9a75807f6ebb9cfb54adaf3c4003fa20517cc (HEAD -> master, origin/master) Add my_m
odule [div&comment]
d4b93dfdae32990616b929ce35eca6f5e0a47259 Add my_module [div&comment]
6f2d5649b035032200b67663e3eb2fee195305d1 Add my_module [div]
295d370a60d0fe4b84aaa4d352861797bc76ea99 Add my_module [mul]
6156f326003ed6de68135443c976c3883519611e Add my_module [sub]
19777c2e3fd9509b26ed6e9f2bd4aab6a40f6cb8 Add my_module [add]
e83abdf06832a299b7e4e6a32eeb0ec785b4e453 Add Text File [document.txt]

(boostcamp) D:\기리\workspace\test\Git-Tutorial>
```

Git log 다루기

옵션 pretty는
원하는 대로 예쁘게 출력

```
Anaconda Prompt (anaconda3)
(boostcamp) D:\기리\workspace\test\Git-Tutorial>git log --pretty=oneline
29b9a75807f6ebb9cfb54adaf3c4003fa20517cc (HEAD -> master, origin/master) Add my_module [div&comment]
d4b93dfdae32990616b929ce35eca6f5e0a47259 Add my_module [div&comment]
6f2d5649b035032200b67663e3eb2fee195305d1 Add my_module [div]
295d370a60d0fe4b84aaa4d352861797bc76ea99 Add my_module [mul]
6156f326003ed6de68135443c976c3883519611e Add my_module [sub]
19777c2e3fd9509b26ed6e9f2bd4aab6a40f6cb8 Add my_module [add]
e83abdf06832a299b7e4e6a32eeb0ec785b4e453 Add Text File [document.txt]

(boostcamp) D:\기리\workspace\test\Git-Tutorial>
```

Git log 다루기

많이 쓰는 거

Git log --pretty=format:"%h -> %an, %ar : %s"--graph

%h : commit의 해시값

%an : 작성자 이름

%ar : 작성 날짜

%s : commit 메시지

```
Anaconda Prompt (anaconda3)
(boostcamp) D:\기리\workspace\test\Git-Tutorial>git log --pretty=format:"%h -> %an, %ar : %s"--graph
n, %ar : %s" --graph
* 29b9a75 -> hee, 26 minutes ago : Add my_module [div&comment]
| \
| * 6f2d564 -> hee, 43 minutes ago : Add my_module [div]
* | d4b93df -> hee, 35 minutes ago : Add my_module [div&comment]
| /
* 295d370 -> hee, 60 minutes ago : Add my_module [mul]
* 6156f32 -> hee, 73 minutes ago : Add my_module [sub]
* 19777c2 -> hee, 82 minutes ago : Add my_module [add]
* e83abdf -> hee, 3 hours ago : Add Text File [document.txt]

(boostcamp) D:\기리\workspace\test\Git-Tutorial>
```

Git에서 특정 commit으로 돌아 갈 때

특정 시점의 commit으로 돌아가고 싶을 때

git reset 명령을 사용해 그 시점의 commit의 hash값을 사용

--hard는 특정 commit 지점 이후를 다 지워버린다.

--soft는 commit 이력을 add상태로 올린다

```
(boostcamp) D:\기리\workspace\test\Git-Tutorial>git reset --hard e83abdf06832a299  
b7e4e6a32eeb0ec785b4e453  
HEAD is now at e83abdf Add Text File [document.txt]  
  
(boostcamp) D:\기리\workspace\test\Git-Tutorial>git log  
commit e83abdf06832a299b7e4e6a32eeb0ec785b4e453 (HEAD -> master)  
Author: hee <r1fgmltkfkd@naver.com>  
Date:   Wed Jan 20 03:25:10 2021 +0900  
  
        Add Text File [document.txt]
```

Git에서 특정 commit으로 돌아 갈 때

특정 시점의 commit으로 돌아가고 싶을 때

git reset 명령을 사용해 그 시점의 commit의 hash값을 사용

--hard는 특정 commit 지점 이후를 다 지워버린다.

--soft는 commit 이력을 add상태로 올린다

<https://www.devpools.kr/2017/02/05/%EC%B4%88%EB%B3%B4%EC%9A%A9-git-%EB%90%98%EB%8F%8C%EB%A6%AC%EA%B8%B0-reset-revert/>

Git에서 특정 commit으로 돌아 갈 때

이렇게 해도 알다 싶이

Remote repository는 변경 사항 없음

그런데 Local과 github의 구성이 완전히 달라져서

Push 하면 오류남

git push -f로 강제 push도 가능

Git에서 commit 메시지 수정하기

Git commit --amend

를 하면 바로 이전 commit의 메시지를 수정 가능

이렇게 실행하면

Unix 에디터로 들어가서 수정할 수 있게 됨

Unix 에디터의 사용법은

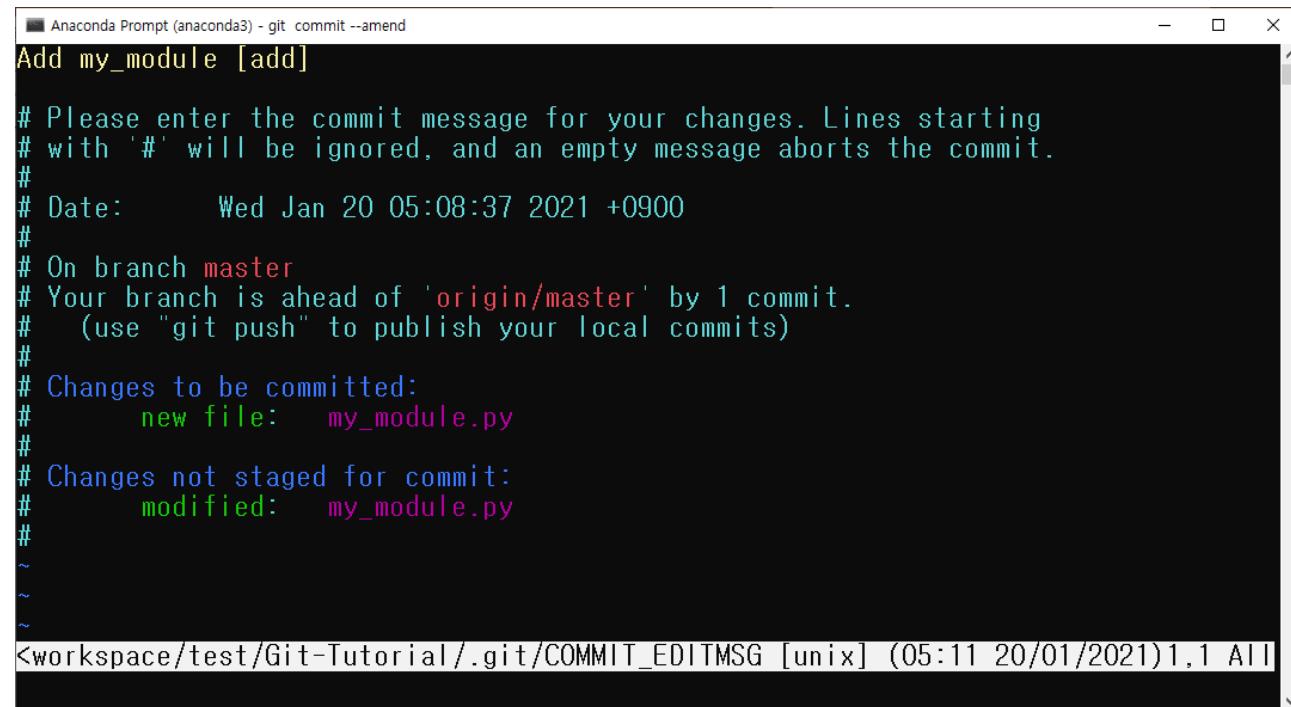
기본적으로 a를 눌러 수정모드가 되며

그 때 값을 수정하고 다 수정하면

ESC로 나오고

:WQ! 에디터 끔

해당 파일이 수정된 내용으로 저장됨



The screenshot shows a terminal window titled "Anaconda Prompt (anaconda3) - git commit --amend". The command "Add my_module [add]" has been entered. The terminal displays the commit message editor interface with the following text:

```
Add my_module [add]

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Wed Jan 20 05:08:37 2021 +0900
#
# On branch master
# Your branch is ahead of 'origin/master' by 1 commit.
#   (use "git push" to publish your local commits)
#
# Changes to be committed:
#       new file:  my_module.py
#
# Changes not staged for commit:
#       modified: my_module.py
#
~
~
~

<workspace/test/Git-Tutorial/.git/COMMIT_EDITMSG [unix] (05:11 20/01/2021) 1,1 All
```

branch

동시에 여러 개발자들이 프로젝트에서 각기 다른 기능을 개발할 수 있도록 하는 기능
서로 다른 branch는 작업을 할 때 서로에게 영향을 받지 않는다는 점에서 마음 놓고 서
로 다른 개발 작업을 수행할 수 있음

Git repository를 만들면 자동으로 master branch 생성

Master branch는 항상 안정화 되어 있도록 한다

개발 branch 와 다른 branch가 따로 있어야

개발 과정에서의 오류가 master branch에 영향을 주지 않는다.

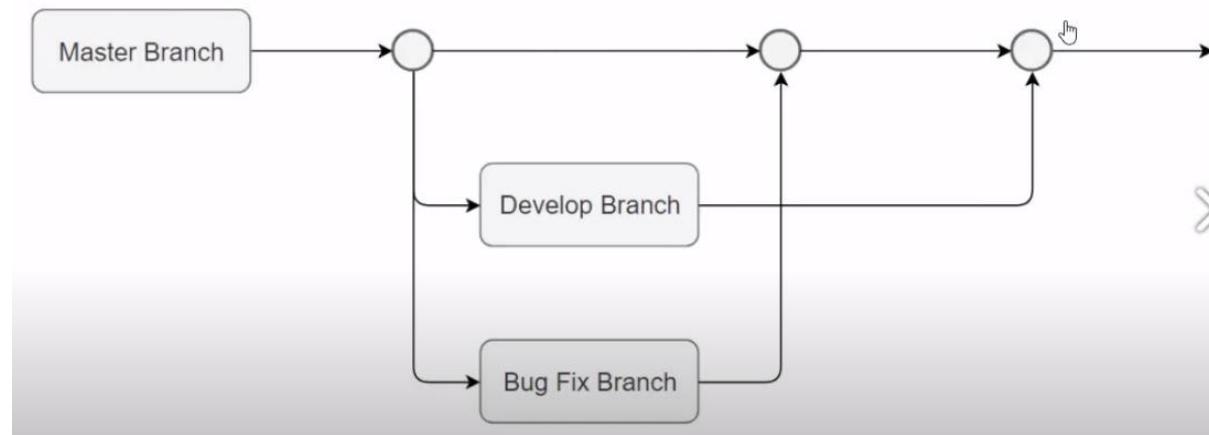
원하는 타이밍에 안정화 뒷을때 master branch에

Merge를 시킴

branch

통합 branch : 배포가 가능한 수준의 branch 일반적으로 master branch

토픽 branch : 특정 기능을 위해 만들어진 branch 일반적으로 master branch 외의 다른 branch들



branch

Git branch 명령어로 현재 브랜치를 알 수 있음

Repository에 존재하는
모든 branch들의 목록과
현재 위치한 branch를 보여줌

```
선택 Anaconda Prompt (anaconda3)

(boostcamp) D:\기리\workspace\test\Git-Tutorial>git branch
* master

(boostcamp) D:\기리\workspace\test\Git-Tutorial>git branch develop
(boostcamp) D:\기리\workspace\test\Git-Tutorial>git branch
  develop
* master

(boostcamp) D:\기리\workspace\test\Git-Tutorial>
```

branch

Branch를 옮기려면
Git checkout branch이름

```
Anaconda Prompt (anaconda3)

(boostcamp) D:\기리\workspace\test\Git-Tutorial>git branch
  develop
* master

(boostcamp) D:\기리\workspace\test\Git-Tutorial>git checkout develop
Switched to branch 'develop'

(boostcamp) D:\기리\workspace\test\Git-Tutorial>git branch
* develop
  master

(boostcamp) D:\기리\workspace\test\Git-Tutorial>
```

branch

Branch가 바뀐 상태에서 소스코드를 변경

그리고 git log를 확인해보면

Head가 develop를 가르키고 있다.

아직은 github에 변화가 없는 상태

```
Anaconda Prompt (anaconda3) - git log
1 file changed, 3 insertions(+)

(boostcamp) D:\기리\workspace\test\Git-Tutorial>git log
commit 295d370a60d0fe4b84aaa4d352861797bc76ea99 (HEAD -> develop)
Author: hee <r1fgmltkfkd@naver.com>
Date:   Wed Jan 20 05:31:04 2021 +0900

    Add my_module [mul]

commit 6156f326003ed6de68135443c976c3883519611e (origin/master, master)
Author: hee <r1fgmltkfkd@naver.com>
Date:   Wed Jan 20 05:18:14 2021 +0900

    Add my_module [sub]

commit 19777c2e3fd9509b26ed6e9f2bd4aab6a40f6cb8
Author: hee <r1fgmltkfkd@naver.com>
Date:   Wed Jan 20 05:08:37 2021 +0900

    Add my_module [add]

commit e83abdf06832a299b7e4e6a32eeb0ec785b4e453
```

branch

Merge를 하려면
우선 master branch로 돌아가고
Merge를 하고
Git log를 출력해보면
마지막 commit이 master에도 반영된 것을 볼 수 있음

```
[Anaconda Prompt (anaconda3)]
(boostcamp) D:\기리\workspace\test\Git-Tutorial>git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

(boostcamp) D:\기리\workspace\test\Git-Tutorial>git branch
  develop
* master

(boostcamp) D:\기리\workspace\test\Git-Tutorial>git merge develop
Updating 6156f32..295d370
Fast-forward
  my_module.py | 3 +++
  1 file changed, 3 insertions(+)
```

```
[Anaconda Prompt (anaconda3) - git log]
1 file changed, 3 insertions(+)

(boostcamp) D:\기리\workspace\test\Git-Tutorial>git log
commit 295d370a60d0fe4b84aaa4d352861797bc76ea99 (HEAD -> master, develop)
Author: hee <r1fgmltkfkd@naver.com>
Date:   Wed Jan 20 05:31:04 2021 +0900

    Add my_module [mul]

commit 6156f326003ed6de68135443c976c3883519611e (origin/master)
Author: hee <r1fgmltkfkd@naver.com>
Date:   Wed Jan 20 05:18:14 2021 +0900

    Add my_module [sub]

commit 19777c2e3fd9509b26ed6e9f2bd4aab6a40f6cb8
Author: hee <r1fgmltkfkd@naver.com>
Date:   Wed Jan 20 05:08:37 2021 +0900

    Add my_module [add]

commit e83abdf06832a299b7e4e6a32eeb0ec785b4e453
```

branch

Merge 하고서 push하면 github에 develop branch 없음

코드 수정은 적용됨

그러나 git branch 하면 develop 남아있다.

쓸모를 다른 branch는

Git branch -d develop

를 통해 제거가 가능하다.

```
(boostcamp) D:\기리\workspace\test\Git-Tutorial>git branch
  develop
* master

(	boostcamp) D:\기리\workspace\test\Git-Tutorial>git branch -d develop
Deleted branch develop (was 295d370).

(	boostcamp) D:\기리\workspace\test\Git-Tutorial>git branch
* master

(	boostcamp) D:\기리\workspace\test\Git-Tutorial>
```

branch

그럼 merge전에 push하면?

오류가 난다.

왜?

해결법은?

```
Anaconda Prompt (anaconda3)
(boostcamp) D:\기리\workspace\test\Git-Tutorial>git checkout develop
Switched to branch 'develop'

(boostcamp) D:\기리\workspace\test\Git-Tutorial>git branch
* develop
  master

(boostcamp) D:\기리\workspace\test\Git-Tutorial>git add .

(boostcamp) D:\기리\workspace\test\Git-Tutorial>git commit -m "Add my_module [div]"
[develop 6f2d564] Add my_module [div]
 1 file changed, 3 insertions(+)

(boostcamp) D:\기리\workspace\test\Git-Tutorial>git push
fatal: The current branch develop has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin develop

(boostcamp) D:\기리\workspace\test\Git-Tutorial>
```

branch

Merge 과정에서 발생할 수 있는 충돌 오류

일반적으로 두개 이상의 branch에서 동일한 파일을 수정을 해서 두 파일이 서로 다를 때 merge를 하면 충돌이 일어날 수 있음

예시 github에는 mul까지 작성된 코드

Develop branc에는 div까지 작성된 코드

Master branc에는 주석까지 달린 코드

branch

예시 github에는 mul까지 작성된 코드
Develop branc에는 div까지 작성된 코드
Master branc에는 주석까지 달린 코드

이 경우 각 branch에서의 git log가 다르게 나옴

```
Anaconda Prompt (anaconda3) - git log

(boostcamp) D:\기리\workspace\test\Git-Tutorial>git branch
  develop
* master

(boostcamp) D:\기리\workspace\test\Git-Tutorial>git log
commit d4b93dfdae32990616b929ce35eca6f5e0a47259 (HEAD -> master)
Author: hee <rlfgmltkfd@naver.com>
Date:   Wed Jan 20 05:56:16 2021 +0900

    Add my_module [div&comment]

commit 295d370a60d0fe4b84aaa4d352861797bc76ea99 (origin/master)
Author: hee <rlfgmltkfd@naver.com>
Date:   Wed Jan 20 05:31:04 2021 +0900

    Add my_module [mul]

commit 6156f326003ed6de68135443c976c3883519611e
Author: hee <rlfgmltkfd@naver.com>
Date:   Wed Jan 20 05:18:14 2021 +0900
```

```
Anaconda Prompt (anaconda3) - git log

(boostcamp) D:\기리\workspace\test\Git-Tutorial>git branch
* develop
  master

(boostcamp) D:\기리\workspace\test\Git-Tutorial>git log
commit 6f2d5649b035032200b67663e3eb2fee195305d1 (HEAD -> develop)
Author: hee <rlfgmltkfd@naver.com>
Date:   Wed Jan 20 05:48:29 2021 +0900

    Add my_module [div]

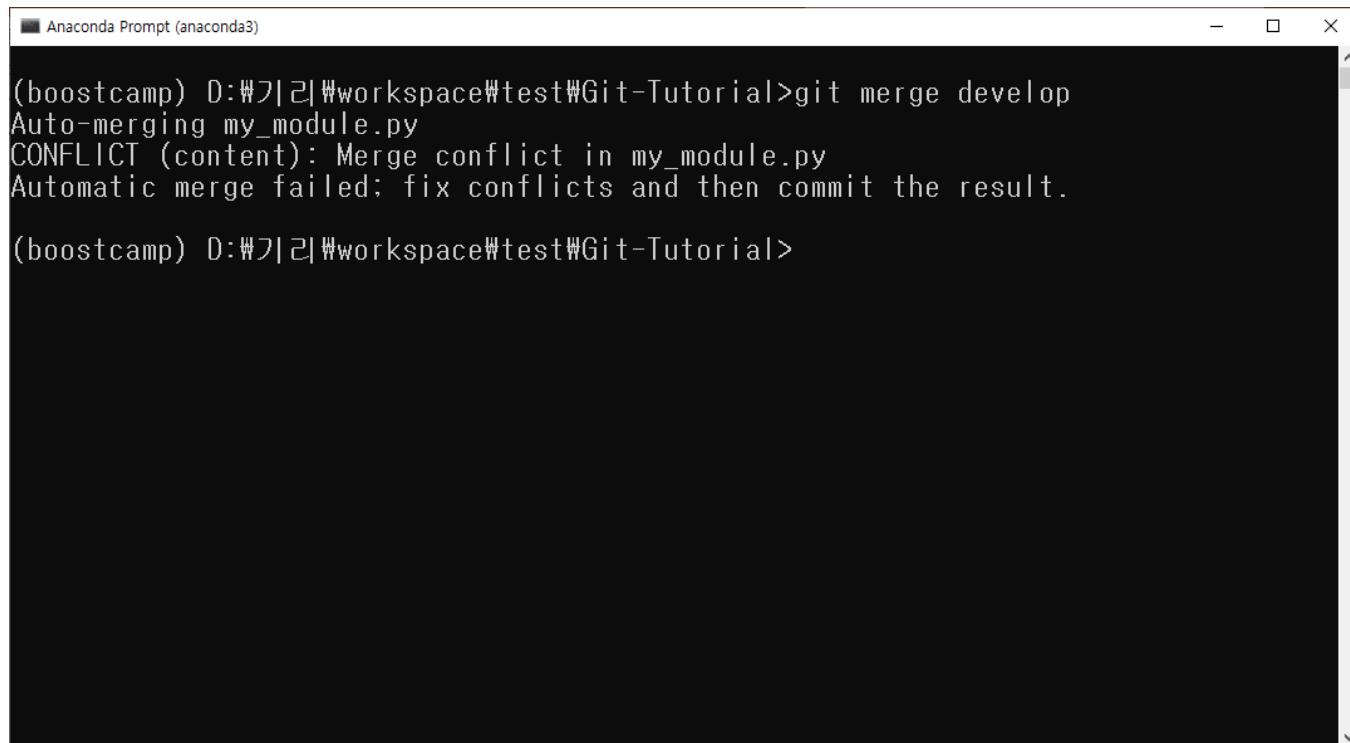
commit 295d370a60d0fe4b84aaa4d352861797bc76ea99 (origin/master)
Author: hee <rlfgmltkfd@naver.com>
Date:   Wed Jan 20 05:31:04 2021 +0900

    Add my_module [mul]

commit 6156f326003ed6de68135443c976c3883519611e
Author: hee <rlfgmltkfd@naver.com>
Date:   Wed Jan 20 05:18:14 2021 +0900
```

branch

이 경우 각 branch에서의 git log가 다르게 나옴
왜냐하면 branch는 서로 같지 않으므로
이 상태에서 merge를 하면 충돌 발생



A screenshot of a terminal window titled "Anaconda Prompt (anaconda3)". The window shows the following command and its output:

```
(boostcamp) D:\기리\workspace\test\Git-Tutorial>git merge develop
Auto-merging my_module.py
CONFLICT (content): Merge conflict in my_module.py
Automatic merge failed; fix conflicts and then commit the result.

(boostcamp) D:\기리\workspace\test\Git-Tutorial>
```

The terminal window has a dark background and light-colored text. It shows a standard git merge command followed by an error message indicating a conflict in the file "my_module.py". The user is prompted to fix the conflicts and then commit the changes.

branch

충돌 발생 에러 확인 후 code에 들어가보면
자동으로 소스코드에서 어디가 다른지 명시되게 변경 되있음

둘 중 어떤걸 결정할지 수동으로 하는 방법

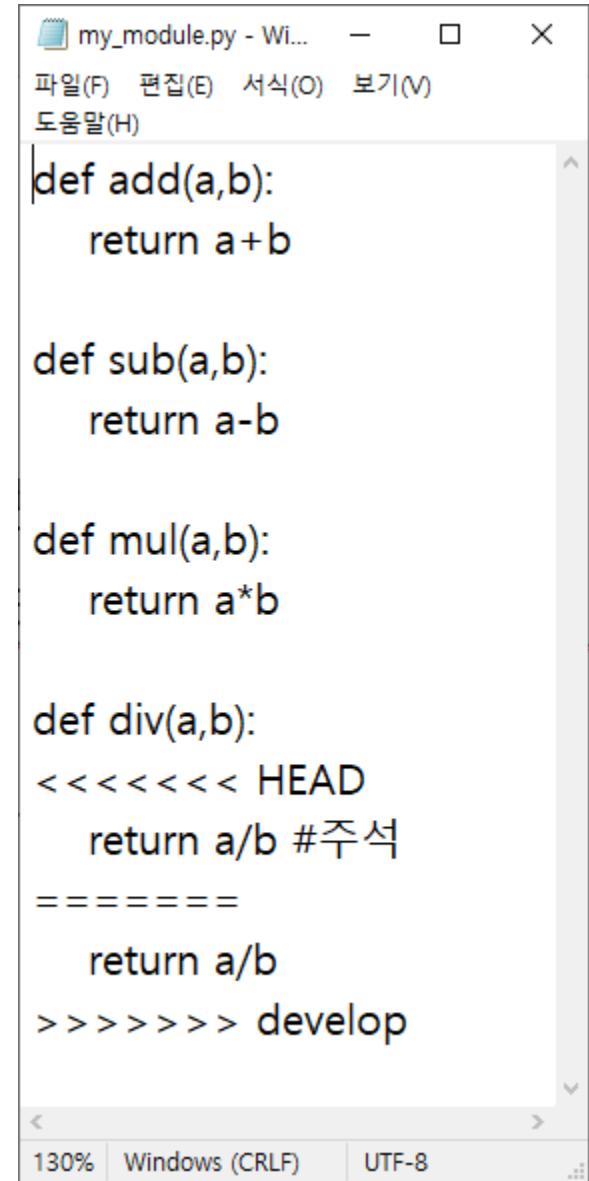
여기서 원하는 걸로 수정해서 저장

그리고 add commit 다시하고

Merge 진행해보면 이미 병합이 되었다고 나옴

즉 이미 새롭게 add commit하면서 둘 중 원하는 코드로 수정된
게 저장되었으니까 merge 됬다고 하는것

Merge되서 이제 git log 하나에 나옴



```
my_module.py - Wi... 파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
def add(a,b):
    return a+b

def sub(a,b):
    return a-b

def mul(a,b):
    return a*b

def div(a,b):
<<<<<< HEAD
    return a/b #주석
=====
    return a/b
>>>>>> develop
```

branch

둘 중 어떤 걸 결정할지 수동으로 하는 방법

여기서 원하는 걸로 수정해서 저장

그리고 add commit 다시하고

Merge 진행해보면 이미 병합이 되었다고 나옴

즉 이미 새롭게 add commit하면서 둘 중 원하는 코드로 수정된

게 저장 됫으니까 merge 됫다고 하는 것

Merge 되서 이제 git log 하나에 나옴

```
Anaconda Prompt (anaconda3) - git log
(bootcamp) D:\기리\workspace\test\Git-Tutorial>git log
commit 29b9a75807f6ebb9cfb54adaf3c4003fa20517cc (HEAD -> master)
Merge: d4b93df 6f2d564
Author: hee <r1fgm1tkfk1d@naver.com>
Date:   Wed Jan 20 06:04:54 2021 +0900

    Add my_module [div&comment]

commit d4b93dfdae32990616b929ce35eca6f5e0a47259
Author: hee <r1fgm1tkfk1d@naver.com>
Date:   Wed Jan 20 05:56:16 2021 +0900

    Add my_module [div&comment]

commit 6f2d5649b035032200b67663e3eb2fee195305d1 (develop)
Author: hee <r1fgm1tkfk1d@naver.com>
Date:   Wed Jan 20 05:48:29 2021 +0900

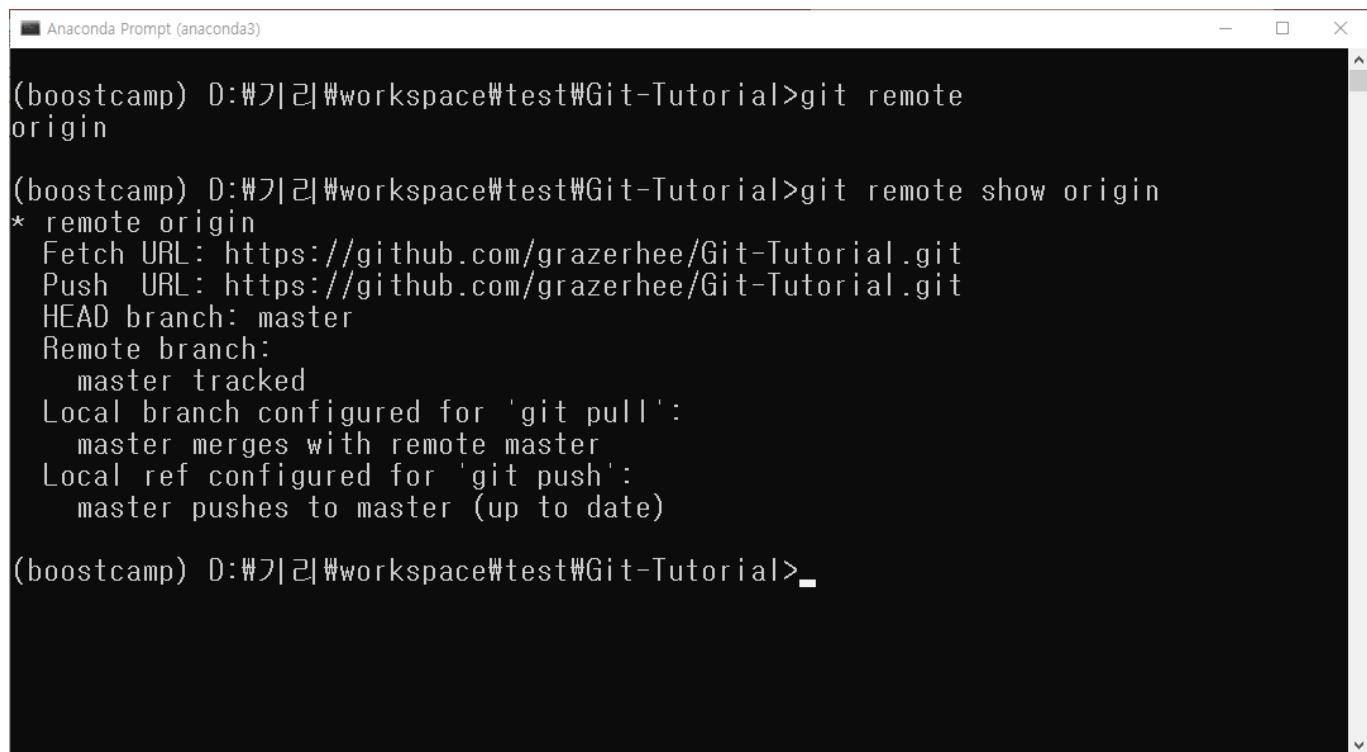
    Add my_module [div]

commit 295d370a60d0fe4b84aaa4d352861797bc76ea99 (origin/master)
```

Git 원격 저장소 관리

Git remote는 현재 원격 저장소가
뭘로 등록 되어있는지 출력

특정 원격 저장소에 대한 정보를
자세히 확인하고 싶으면
Git remote show origin



Anaconda Prompt (anaconda3)

```
(boostcamp) D:\기리\workspace\test\Git-Tutorial>git remote origin
* remote origin
  Fetch URL: https://github.com/grazerhee/Git-Tutorial.git
  Push URL: https://github.com/grazerhee/Git-Tutorial.git
  HEAD branch: master
  Remote branch:
    master tracked
  Local branch configured for 'git pull':
    master merges with remote master
  Local ref configured for 'git push':
    master pushes to master (up to date)

(bootcamp) D:\기리\workspace\test\Git-Tutorial>
```

Git 원격 저장소 관리

원격 저장소 등록

Git remote add 이름 url

Git remote -v하면 현재 등록된 원격 저장소 목록을 보여줌

원격 저장소 이름 변경

Git remote rename a b a를 b로 변경

등록하고 나면 git의 다양한 명령어를 특정 저장소에 대해서 수행할 수도 있음

Git 원격 저장소 관리

원격 저장소 삭제

Git remote rm 저장소이름

물론 일반적으로 프로젝트 하나에 원격 저장소 하나를 사용하는 것이 일반적이지만 이런 방법도 있다 정도

Git에서 README.md 파일 작성

이건 그냥 github에서 작성하는 것이 가장 편하고
그 후에 pull해서 local로 받아온다.

혹시 필요하다면...

<https://www.youtube.com/watch?v=MFJIOqxK6k8&list=PLRx0vPvlEmdD5FLIdwTM4mKBgyjv4no81&index=11>

직접 보세요...

Git Archive

깃 아카이브는 깃 프로젝트에서 소스코드만 추출하고 싶을 때 사용하는 명령

```
Git archive --format=zip master -o Master.zip
```

Branch를 설정

만들어질 파일명/ 만들어질 파일 위치까지

이렇게 하면 그 폴더 내에 원하는 압축파일이 생성됨



A screenshot of the Anaconda Prompt window titled "Anaconda Prompt (anaconda3)". The command line shows the execution of the "git archive" command:

```
(boostcamp) D:\기리\workspace\test\Git-Tutorial>git archive --format=zip master -o Master.zip
```

The command is completed successfully, indicated by the prompt "(boostcamp) D:\기리\workspace\test\Git-Tutorial>" at the bottom.

Git Rebase

특정한 커밋을 수정/삭제
커밋의 log가 이렇다고 할때
이중 delete example1
커밋을 수정하고자하면?
앞선 git commit --amend는
직전 commit만 수정이 가능하므로
Git rebase를 사용 가능

```
Anaconda Prompt (anaconda3) - git log
commit 29b9a75807f6ebb9cfb54adaf3c4003fa20517cc (origin/master)
(boostcamp) D:\#기리\workspace\test\Git-Tutorial>git log
commit c0f9bf44399d73221dee0fd9989431f826c0db4d (HEAD -> master)
Author: hee <r1fgmltkfkd@naver.com>
Date:   Wed Jan 20 06:54:36 2021 +0900

    add example2

commit bb24d7c0550f3fc56aebd63325e6813374068907
Author: hee <r1fgmltkfkd@naver.com>
Date:   Wed Jan 20 06:53:59 2021 +0900

    delete example1

commit be2bd5aa96ee9d007e14ad03f050b359070c2643
Author: hee <r1fgmltkfkd@naver.com>
Date:   Wed Jan 20 06:53:27 2021 +0900

    update example1

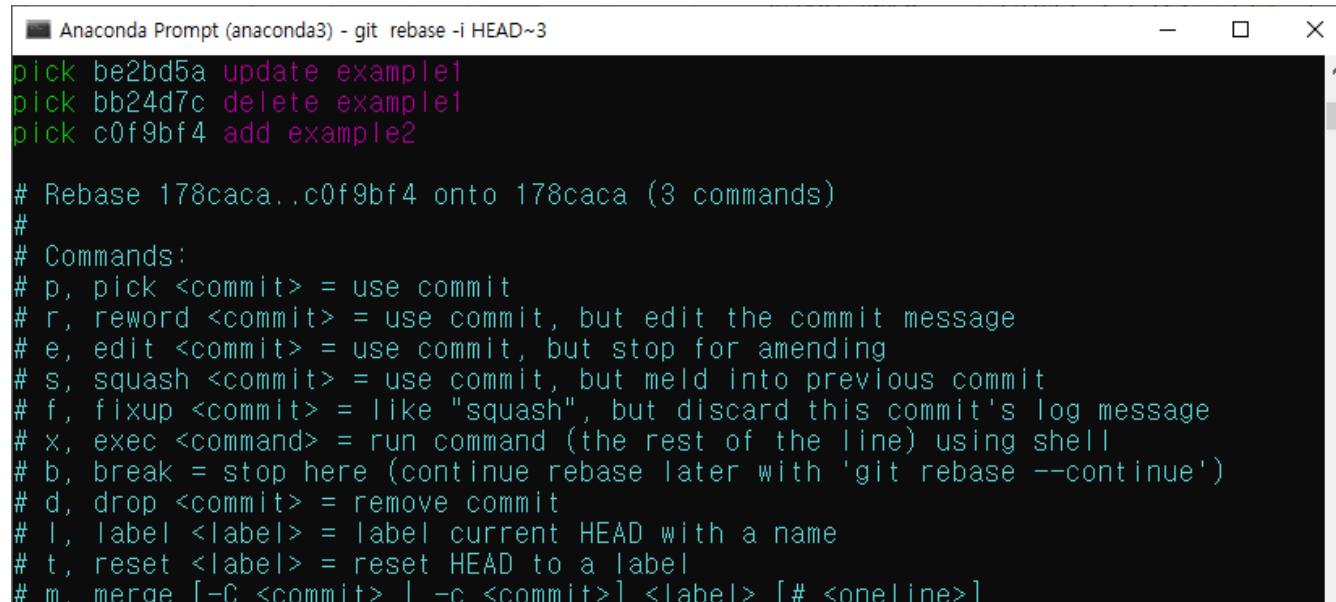
commit 178caca606cbeef9fb4fc107f1ffc85bb7cea562
Author: hee <r1fgmltkfkd@naver.com>
Date:   Wed Jan 20 06:52:38 2021 +0900

    add example1
```

Git Rebase

Git rebase -i HEAD~3

-i 인터랙티브 모드 어떤 커밋 내용을 어떻게 수정할 것인지 편하게 가능
HEAD~3 헤드를 기준으로 3개의 커밋을 연다.



```
Anaconda Prompt (anaconda3) - git rebase -i HEAD~3
pick be2bd5a update example1
pick bb24d7c delete example1
pick c0f9bf4 add example2

# Rebase 178caca..c0f9bf4 onto 178caca (3 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup <commit> = like "squash", but discard this commit's log message
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit>] [-c <commit>] <label> [<oneline>]
```

Git Rebase

각 커밋 앞의 pick의 자리에 밑의 command를 참고하여 원하는 기능에 맞게 변경
수정은 아까 unix와 동일

A로 수정모드 들어가서 변경하고 esc로 나와 :wq!로 저장후 나가기

하면 다음과 같이 나옴

이 안에서 원하는 대로

커밋 메시지를 수정하고

또 똑같이 나온다.

Git Rebase

```
Anaconda Prompt (anaconda3) - git log
commit ebcffc54d77baf2d4d422aab26ccdf0de0ce4a87 (HEAD -> master)
Author: hee <rifgmltkfkd@naver.com>
Date:   Wed Jan 20 06:54:36 2021 +0900

    add example2

commit ed88f9238ba92fde050b8a893f39e74f655fffd48
Author: hee <rifgmltkfkd@naver.com>
Date:   Wed Jan 20 06:53:59 2021 +0900

    drop example1

commit be2bd5aa96ee9d007e14ad03f050b359070c2643
Author: hee <rifgmltkfkd@naver.com>
Date:   Wed Jan 20 06:53:27 2021 +0900

    update example1

commit 178caca606cbeef9fb4fc107f1ffc85bb7cea562
Author: hee <rifgmltkfkd@naver.com>
Date:   Wed Jan 20 06:52:38 2021 +0900

    add example1

commit 29b9a75807f6ebb9cfb54adaf3c4003fa20517cc (origin/master)
:
```

결과
Drop으로 바뀜

Git Rebase

오래된 커밋에 대해서도
Git rebase -i 커밋의 hash
를 해주면 해당 커밋에 대한 메시지 수정이 가능하다.

Git Rebase

중간에 있는 커밋 자체를 빼고 싶으면?

이때도 git rebase 사용

충돌 발생할 수 있으므로 권장하지는 않음

이땐 pick을 drop으로 하면 됨

지워진 것을 확인할 수 있음

```
Anaconda Prompt (anaconda3) - git rebase -i HEAD~3
pick be2bd5a update example1
drop ed88f92 drop example1
pick ebcfffc5 add example2

# Rebase 178caca..ebcfffc5 onto 178caca (3 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup <commit> = like "squash", but discard this commit's log message
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit>] [-c <commit>] <label> [<oneline>]
# .      create a merge commit using the original merge commit's
# .      message (or the oneline, if no original merge commit was
# .      specified). Use -c <commit> to reword the commit message.
#
# These lines can be re-ordered; they are executed from top to bottom.
#
<t-Tutorial/.git/rebase-merge/git-rebase-todo[+] [unix] (07:08 20/01/2021)2,4 Top
:wq!
```

Git Rebase

```
■ Anaconda Prompt (anaconda3) - git log
commit 5f0f30df90bc3b648e8dfefc0c3ba58fc745485 (HEAD -> master)
Author: hee <r1fgmltkfkd@naver.com>
Date:   Wed Jan 20 06:54:36 2021 +0900

    add example2

commit be2bd5aa96ee9d007e14ad03f050b859070c2643
Author: hee <r1fgmltkfkd@naver.com>
Date:   Wed Jan 20 06:53:27 2021 +0900

    update example1

commit 178caca606cbeef9fb4fc107f1ffc85bb7cea562
Author: hee <r1fgmltkfkd@naver.com>
Date:   Wed Jan 20 06:52:38 2021 +0900

    add example1

commit 29b9a75807f6ebb9cfb54adaf3c4003fa20517cc (origin/master)
Merge: d4b93df 6f2d564
Author: hee <r1fgmltkfkd@naver.com>
Date:   Wed Jan 20 06:04:54 2021 +0900

    Add my_module [div&comment]
:
```

Drop example1이 사라짐
Local repository에도
Example.txt 파일이 삭제 되지 않아
다시 생성됨

Git Rebase

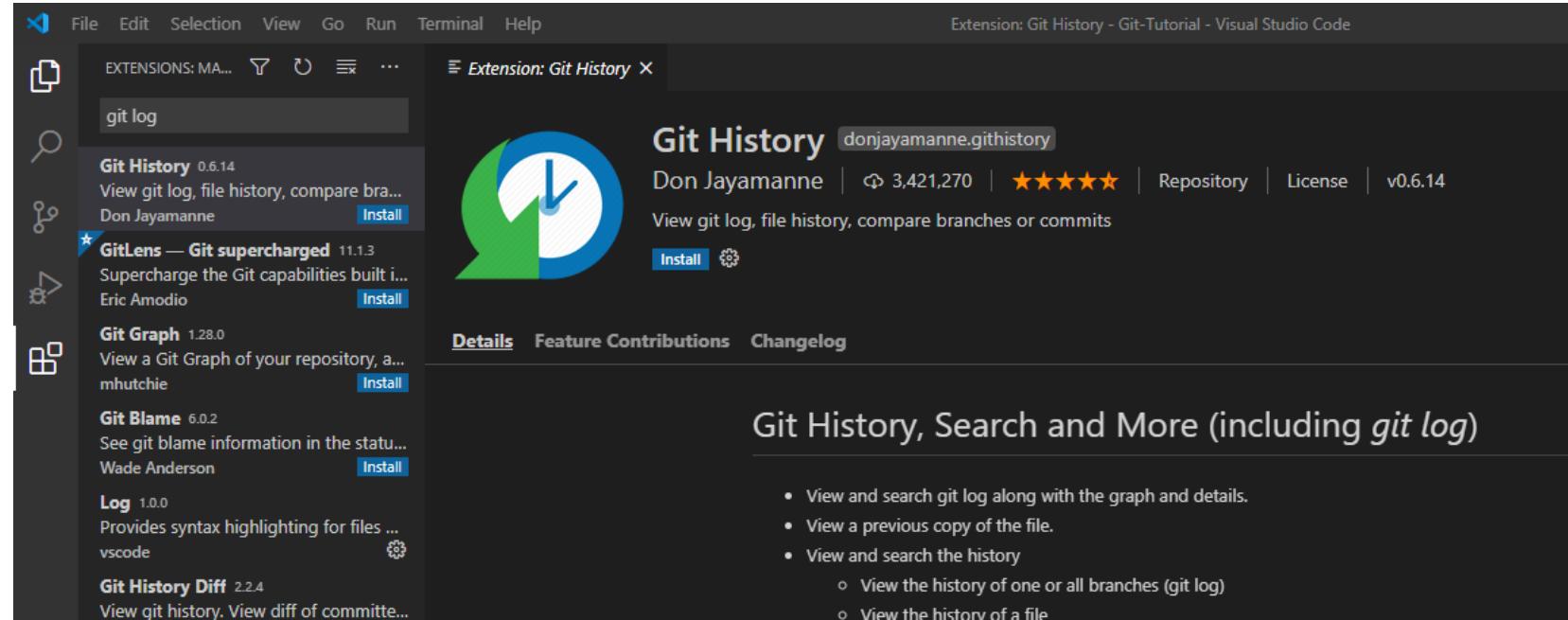
깊게 들어가면 git rebase도 다양한 경우를 처리할 수 있다고 한다...

Merge와 rebase 비교

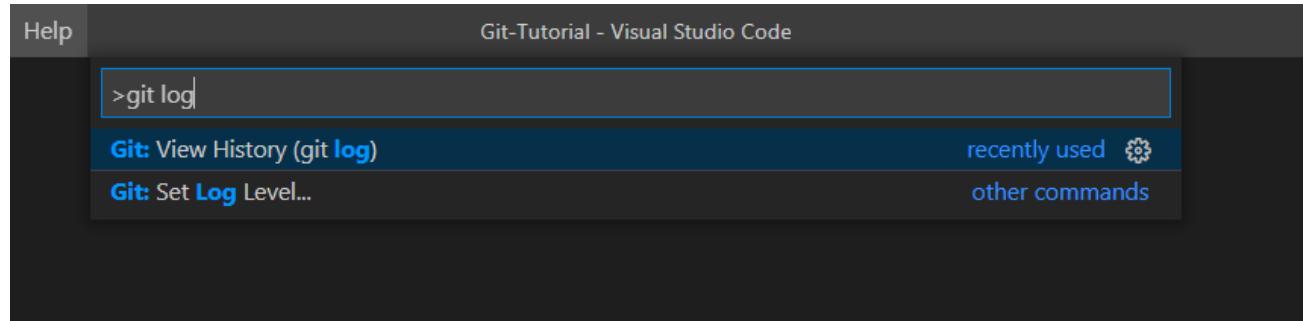
<https://git-scm.com/book/ko/v2/Git-%EB%B8%8C%EB%9E%9C%EC%B9%98-Rebase-%ED%95%98%EA%B8%B0>

VSCode 에서 Git

이 extension을 사용하면 git log 보기 편함



VSCode 에서 Git



A screenshot of the Git History view in VSCode. The title bar says "Git History (Git-Tutorial)". The main area shows a list of commits:

- update example1
hee on 1/20/2021, 6:53:27 AM
Commit ID: be2bd5a
- add example1
hee on 1/20/2021, 6:52:38 AM
Commit ID: 178caca
- Add my_module [div&comment]
hee on 1/20/2021, 6:04:54 AM
Commit ID: 29b9a75
- Add my_module [div&comment]
hee on 1/20/2021, 5:56:16 AM
Commit ID: d4b93df
- Add my_module [div]
hee on 1/20/2021, 5:48:29 AM
Commit ID: 6f2d564
- Add my_module [mul]
hee on 1/20/2021, 5:31:04 AM
Commit ID: 295d370
- Add my_module [sub]
hee on 1/20/2021, 5:18:14 AM
Commit ID: 6156f32
- Add my_module [add]

Each commit entry includes a context menu icon (three dots) and a detailed commit info button.

VSCode 에서 Git

The screenshot shows the VSCode interface with several windows open:

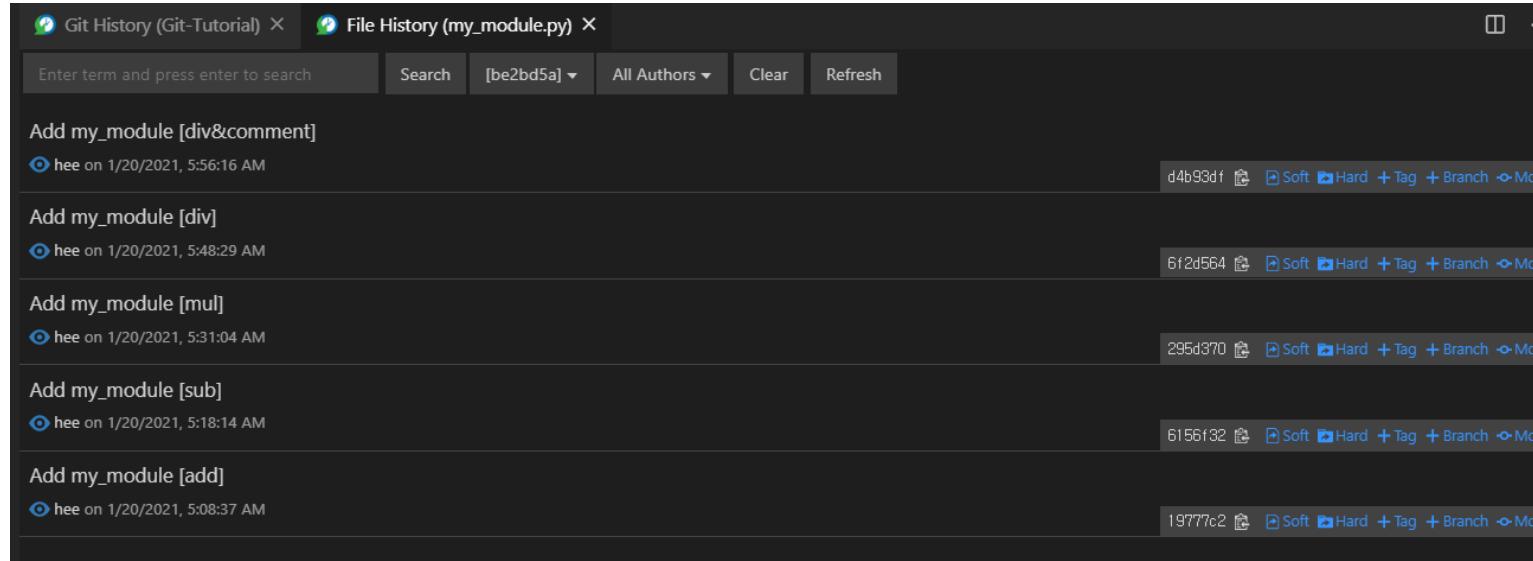
- Git History (Git-Tutorial)**: Shows a list of commits for a repository named "Git-Tutorial".
 - update example1 (commit 1be2bd5a)
 - add example1 (commit 1f78caca)
 - Add my_module [div&comment] (commit 29b9a75)
 - Add my_module [div&comment] (commit 3e03a2d)
 - Add my_module [div] (commit 6f2d564)
 - update example1 (commit 4d4b93df)
- File History (my_module.py)**: Shows a list of file history entries for "my_module.py".
 - Add my_module [div&comment] (commit 1be2bd5a)
 - Add my_module [div] (commit 6f2d564)
 - Add my_module [mul] (commit 295d370)
 - Add my_module [sub] (commit 6155f32)
 - Add my_module [add] (commit 19777c2)
- Code Editor (my_module.py)**: Displays the Python code for "my_module.py". The code defines four functions: add, sub, mul, and div. The div function is currently being edited.

```
update example1
add example1
Add my_module [div&comment]
Add my_module [div&comment]
Add my_module [div]
update example1
example1.txt

Git History (Git-Tutorial)
File History (my_module.py)

C: > Users > rlgm > AppData > Local > Temp > d4b
1 def add(a,b):
2     return a+b
3
4 def sub(a,b):
5     return a-b
6
7 def mul(a,b):
8     return a*b
9
10+def div(a,b):
11+    return a/b #주석
12+
```

VSCode 에서 Git



Reference

- <https://www.youtube.com/watch?v=rhP5pseOJc0>
- <https://www.daleseo.com/git-init/>

unittest (단위 테스트)

테스트와 단위테스트

- 테스트: 소프트웨어가 요구사항에 의해 개발된 산출물이 요구사항과 부합하는지 여부를 검증하기 위한 작업
- 단위 테스트: 모듈 또는 응용 프로그램 내의 개별 코드 단위가 예상대로 작동하는지 확인하는 반복 가능한 활동
- unittest: Python에 포함된 표준 라이브러리, 다양한 테스트를 자동화할 수 있는 기능이 포함되어 있음

unittest에 포함된 주요 개념

- **TestCase** : unittest 프레임 워크의 테스트 조직의 기본 단위
- **Fixture** : 테스트함수의 전 또는 후에 실행되는 함수
 - setUp(): 테스트함수 호출 전, tearDown(): 테스트함수 호출 후
 - 테스트가 실행되기 전에 데이터베이스 테이블을 만들거나 테스트 환경이 예상 된 상태에 있는지 확인하는 데 사용
 - 테스트 후에 사용한 리소스를 정리하거나 결과를 출력하는데 사용
- **Assertion** : 테스트 통과 여부 결정
 - bool test, 객체의 적합성, 적절한 예외 발생 등 다양한 점검을 할 수 있음

unittest 모듈 사용방법

1. import unittest
2. unittest.TestCase를 상속한 테스트 클래스를 작성
3. 테스트 메소드 작성
 - 이름은 `test_`로 시작
 - assert 메소드 사용
 - 알파벳 순으로 실행됨
4. 실행방법
 - 테스트 파일에 아래 코드 추가한 후 실행

```
if __name__ == '__main__':
    unittest.main()
```
 - `python -m unittest` 테스트파일명 (.py 안 붙음)

unittest 모듈 실행 결과

Error	Fail	OK
<pre>.....E ===== ERROR: test_main ----- Ran 9 tests in 0.117s FAILED (errors=1)</pre>	<pre>...F..... ===== FAIL: test_is_between_100_and_999 ----- Traceback (most recent call last): File "/Users/bomi/Desktop/BoostC/test_baseball_game.py", line 28, self.assertEqual(False, bg.is_ AssertionError: False != True ----- Ran 9 tests in 0.571s FAILED (failures=1)</pre>	<pre>.....- ----- Ran 9 tests in 0.655s OK</pre>

- 실행할 때 `-v` 옵션 추가하면 결과 더 상세하게 나옴

```
(base) macaron:baseball-BomiChoi bomi$ python -m unittest -v test_baseball_game
test_get_not_duplicated_three_digit_number (test_baseball_game.TestBaseballGame) ... ok
test_get_strikes_or_ball (test_baseball_game.TestBaseballGame) ... ok
test_is_between_100_and_999 (test_baseball_game.TestBaseballGame) ... ok
test_is_digit (test_baseball_game.TestBaseballGame) ... ok
test_is_duplicated_number (test_baseball_game.TestBaseballGame) ... ok
test_is_no (test_baseball_game.TestBaseballGame) ... ok
test_is_validated_number (test_baseball_game.TestBaseballGame) ... ok
test_is_yes (test_baseball_game.TestBaseballGame) ... ok
test_main (test_baseball_game.TestBaseballGame) ... ok
-----
Ran 9 tests in 0.651s
OK
```

unittest.TestCase의 주요 메소드들

- `setUp()`: 테스트 메서드를 호출하기 바로 직전에 호출되는 메소드 (Fixture)
- `tearDown()`: 테스트 메서드가 끝나고 결과가 기록되고 나서 바로 다음에 호출되는 메소드 (Fixture)
- `run(result=None)`: 테스트를 실행하고, `result` 인자로 전달된 `TestResult`에 결과를 수집
- `skipTest(reason)`: 현재 테스트 건너뜀
- `subTest(msg=None, **params)`: 부분 테스트
- `assert` 메소드

assert 메소드 리스트

Method	확인하는 것
assertEqual(a, b)	$a == b$
assertNotEqual(a, b)	$a != b$
assertTrue(x)	$\text{bool}(x)$ is True
assertFalse(x)	$\text{bool}(x)$ is False
assertRaises(exc, fun, *args, **kwds)	<code>fun(*args, **kwds)</code> raises exc

→ 이 외에도 엄청나게 많은 메소드들이 있음

unittest.mock

- mocking: 테스트 대상 시스템의 일부를 모의 객체로 교체
- patch(): 특정 범위 내에서만 mocking이 가능하도록 해줌

```
from unittest import TestCase, main
from unittest.mock import patch

def hello():
    return "Hello!"

class TestMe(TestCase):
    @patch("__main__.hello", return_value="Mock!")
    def test_hello(self, mock_hello):
        self.assertEqual(hello(), "Mock!")
        self.assertIs(hello, mock_hello)
        mock_hello.assert_called_once_with()

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

```
$ python test_me.py
```

```
.
```

```
Ran 1 test in 0.000s
```

```
OK
```

Reference

- <https://docs.python.org/ko/3/library/unittest.html>
- <http://labs.brandi.co.kr/2018/06/07/kwakjs.html>
- <https://wikidocs.net/16107>
- <https://docs.python.org/ko/3/library/unittest.mock.html>
- <https://www.daleseo.com/python-unittest-mock-patch/>