!목표

1. 코드의 재사용

2. 다른 사람은 코드를 몰라도 되도록

! 달성방법

획일화 , 부분화 , 다양한 용례가 가능하도록 한다.

! 다양한 용례를 사용하기 위한 state의 도입

function
▶ output : 조작범위가 거의 없음

function + state

▶ state + output

output(state)

! git 의 개념 :

dependancy를 공유하지 않는 독립적인 코드저장소

분산버전 컨트롤을 도모하되 sync만 맞추려는 목적이다.

commit 각 단위

! 예제
4칙연산자를 모듈화해서 만들어서
pull push pull-request merge를 만들어준다.

1. __main__.py adder 구현:
2. 절차지향 프로그래밍으로 간단하게 구현 : git commit push
(with git config, git remote githubrepo)

!. adder를 모듈화

main과 module의 분리

module 별로 if __name__ == '__main__':
논리 사용: 스스로 돌아갈 때만 name = main 이기 때문에, 스스로 돌릴 때만 돌아가는 스크립트! 여기에는 주로 testcode를 작성한다.

!. 이걸 git으로 반영하기

unstage (아직 추적하고 있지 않음)

stage (내 로컬 저장소에 올릴 후보 리스트)

commit (local 저장소)

!. branching

master > split > to inf module
> change name just change name

!. three way merge problem >>> conflict resolve !!! : cannot automatically conflict resolve. >>> must one dev handle one!!! to prevent conflict

!. branching management

head of master >>> only check master

head of add >>> only check add

head of inf >>> only check inf

! more explanation : where is the pyhon modules???

before release: >>> you should git, source cord repo >>> adequate build

after release: >>> with assigned installer, install, extracting, usualy corresponding in Linux : PYTHONPATH >>>

a. current dir
b. python path lib
c. python env lib dir

ex: