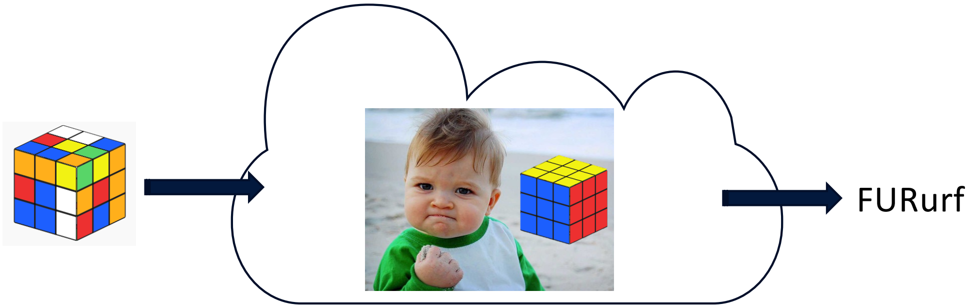


Engineer a microservice that solves Rubik's Cube

Project Vector:

Increment 0 → set up dev environment, customize about

Increment 1 → model cube rotations

Increment 2 → solve "down" cross

Increment 3 → solve lower layer

Increment 4 → solve middle layer

Increment 5 → solve upper surface
add integrity check

Increment 6 → solve top layer
provide error detection (grad students)

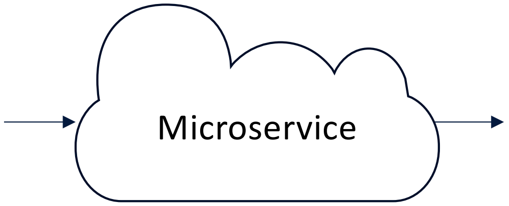
You are here

increment 2

Solve cube's down-face cross

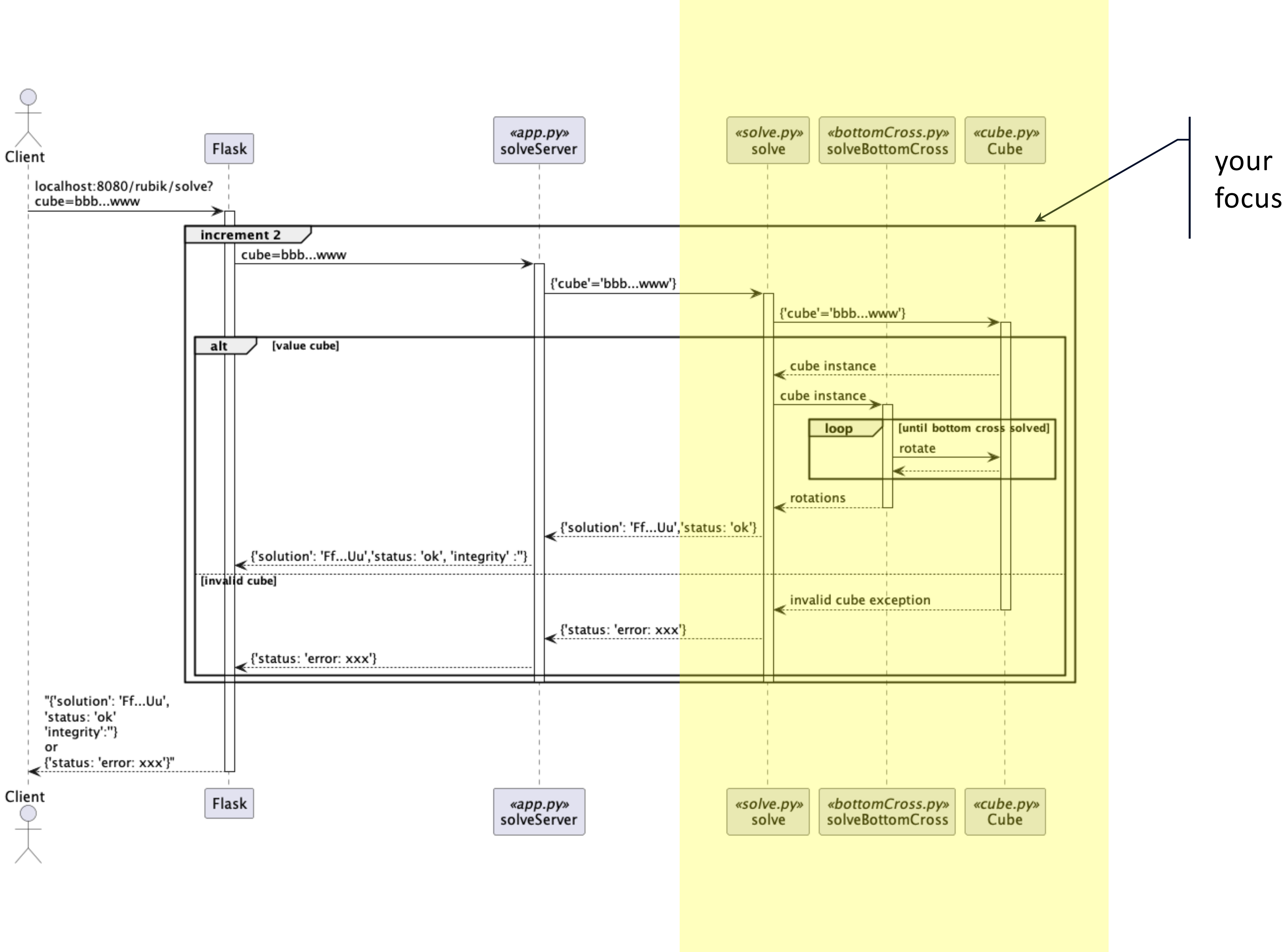
Product Aspect

localhost:8080/rubik/solve?cube=gbbgbbgbbrr
orrorrobggbggbggrooroorooowwywywwwwy
yywywyyyyw


Microservice

{'solution’: RRFFLLBB',
'status': 'ok',
'integrity':''}

output the rotations to perform on the input cube to solve the edge elements of the "down" face.

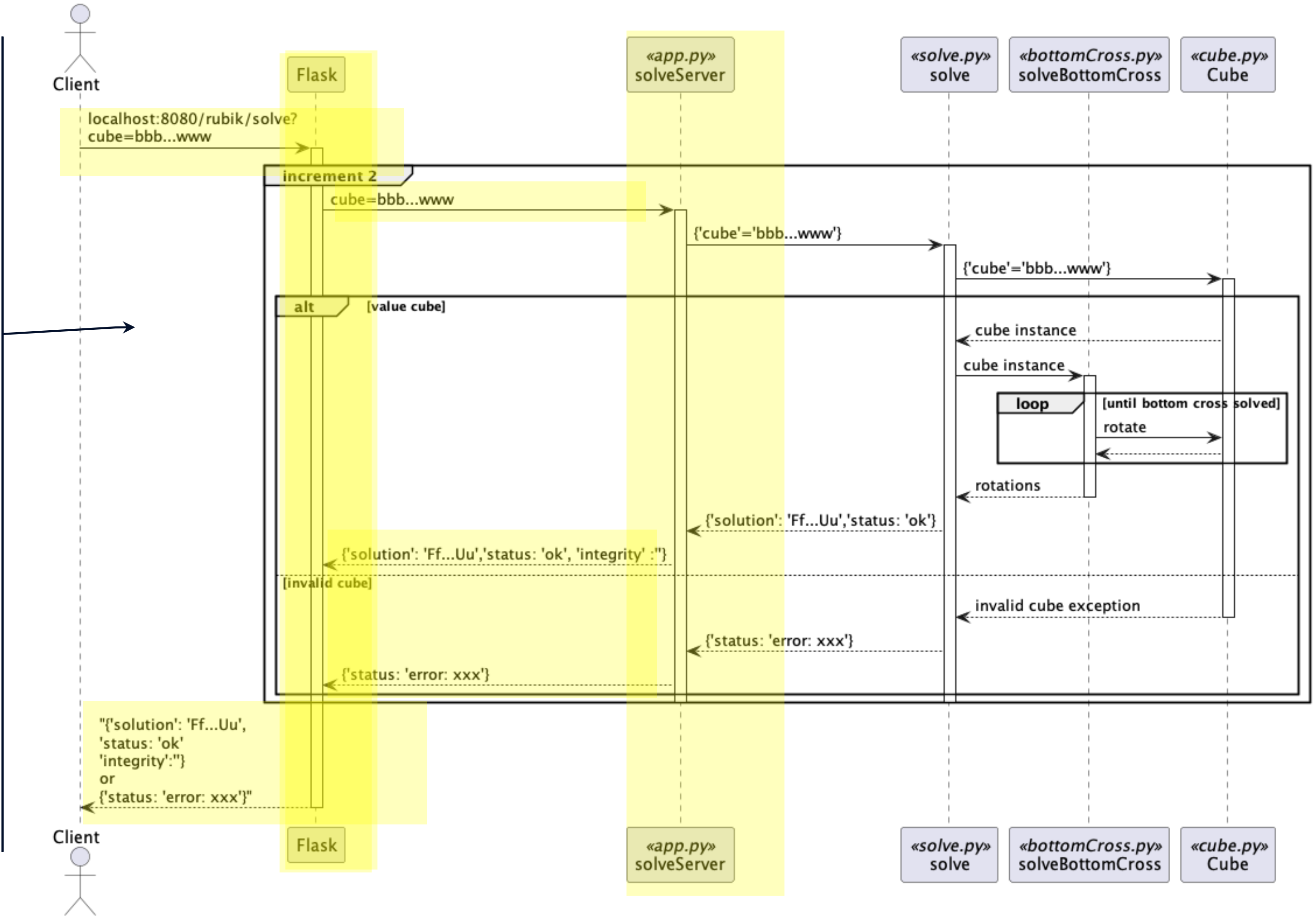

Client
Flask
«app.py»
solveServer
«solve.py»
solve
«bottomCross.py»
solveBottomCross
«cube.py»
Cube
localhost:8080/rubik/solve?
cube=bbb...www
increment 2
cube=bbb...www
{'cube'='bbb...www'}
{'cube'='bbb...www'}
alt
[value cube]
cube instance
cube instance
loop
[until bottom cross solved]
rotate
rotations
{'solution': 'Ff...Uu','status: 'ok'}
{'solution': 'Ff...Uu','status: 'ok', 'integrity' :''}
[invalid cube]
invalid cube exception
{'status: 'error: xxx'}
{'status: 'error: xxx'}
"{'solution': 'Ff...Uu',
'status: 'ok'
'integrity':''}
or
{'status: 'error: xxx'}"
Client
Flask
«app.py»
solveServer
«solve.py»
solve
«bottomCross.py»
solveBottomCross
«cube.py»
Cube

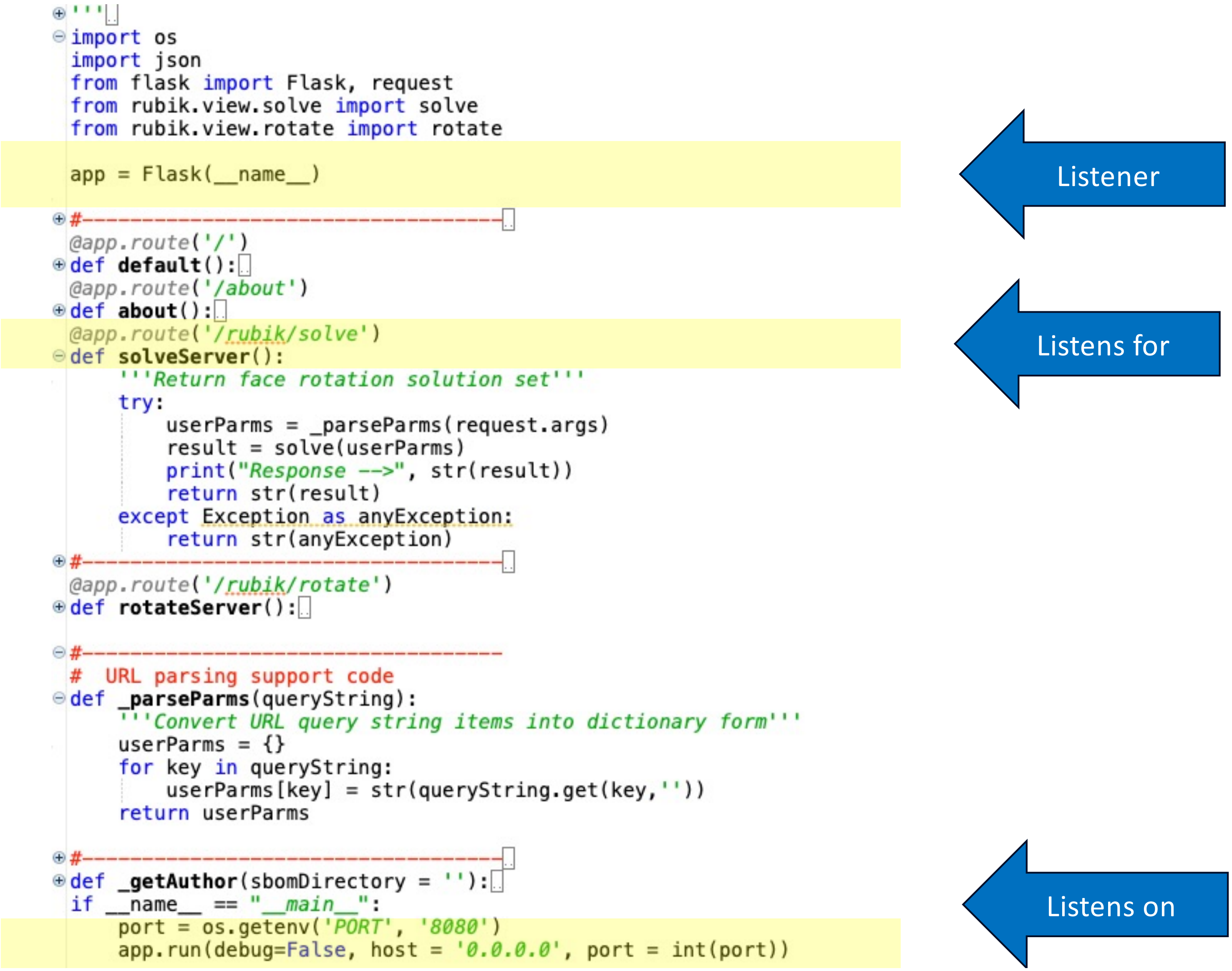
your
focus

Flask listens for an HTTP/HTTPS request of "rubik/solve".

It passes the query string portion of the URL to solveServer(), located in app.py

It places the value returned from solveServer() into the body of the HTTP response





```
'''
import os
import json
from flask import Flask, request
from rubik.view.solve import solve
from rubik.view.rotate import rotate

app = Flask(__name__)

#-----------------------------------
@app.route('/')
def default():
@app.route('/about')
def about():
@app.route('/rubik/solve')
def solveServer():
    '''Return face rotation solution set'''
    try:
        userParms = _parseParms(request.args)
        result = solve(userParms)
        print("Response -->", str(result))
        return str(result)
    except Exception as anyException:
        return str(anyException)
#-----------------------------------
@app.route('/rubik/rotate')
def rotateServer():

#-----------------------------------
#  URL parsing support code
def _parseParms(queryString):
    '''Convert URL query string items into dictionary form'''
    userParms = {}
    for key in queryString:
        userParms[key] = str(queryString.get(key,''))
    return userParms

#-----------------------------------
def _getAuthor(sbomDirectory = ''):
if __name__ == "__main__":
    port = os.getenv('PORT', '8080')
    app.run(debug=False, host = '0.0.0.0', port = int(port))
```

Listener

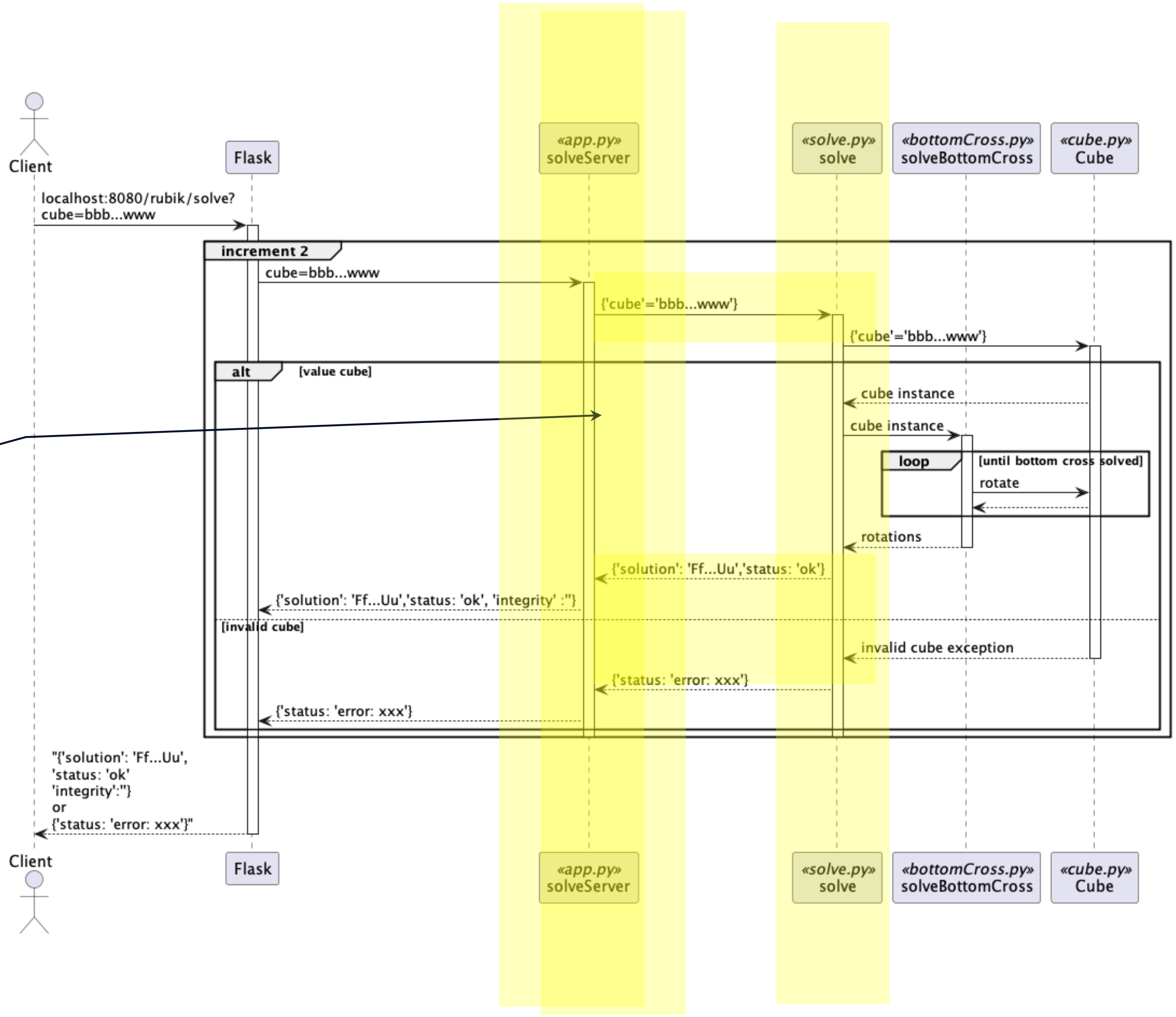
Listens for

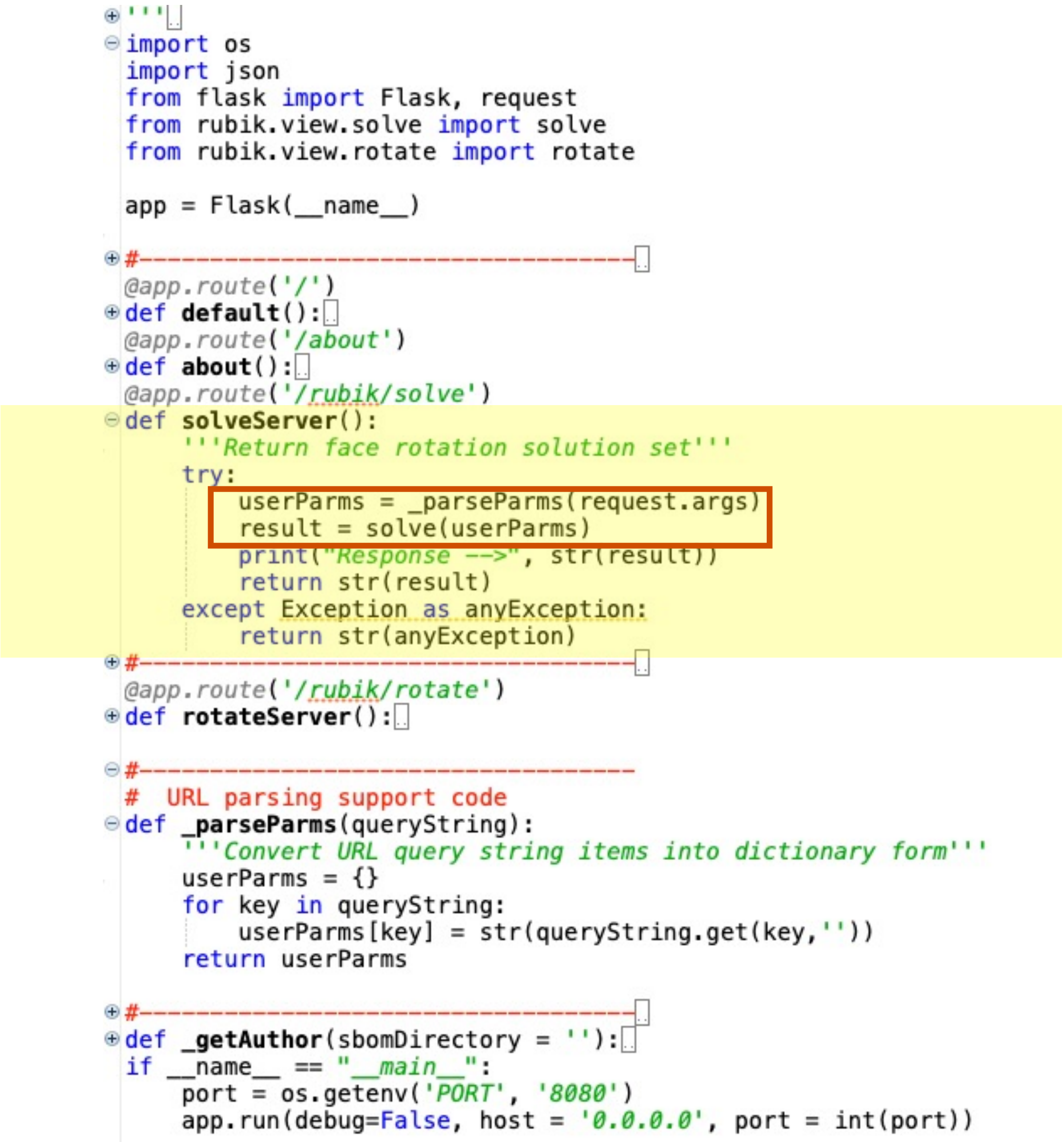
Listens on

solveServer parses the URL query string, converting each element to dictionary format.

It passes this information to solve() (located in solve.py)

It converts the dictionary it receives from solve() to JSON format





```
'''
import os
import json
from flask import Flask, request
from rubik.view.solve import solve
from rubik.view.rotate import rotate

app = Flask(__name__)

#-----------------------------------
@app.route('/')
def default():
@app.route('/about')
def about():
@app.route('/rubik/solve')
def solveServer():
    '''Return face rotation solution set'''
    try:
        userParms = _parseParms(request.args)
        result = solve(userParms)
        print("Response -->", str(result))
        return str(result)
    except Exception as anyException:
        return str(anyException)
#-----------------------------------
@app.route('/rubik/rotate')
def rotateServer():

#-----------------------------------
#  URL parsing support code
def _parseParms(queryString):
    '''Convert URL query string items into dictionary form'''
    userParms = {}
    for key in queryString:
        userParms[key] = str(queryString.get(key,''))
    return userParms

#-----------------------------------
def _getAuthor(sbomDirectory = ''):
if __name__ == "__main__":
    port = os.getenv('PORT', '8080')
    app.run(debug=False, host = '0.0.0.0', port = int(port))
```

1. This code extracts everything to the right of "?" and stores it in a Python dictionary.

For example,
?cube=abc
is converted to
{'cube':'abc'}

2. The dictionary is passed to solve().

3 (happy). The result of solve() is converted to a JSON string and returned to Flask.

3 (sad). If solve() raises an exception, the text portion of the exception is returned to Flask.

1. solve() is passed the parsed query string

2. solve() instantiates the cube

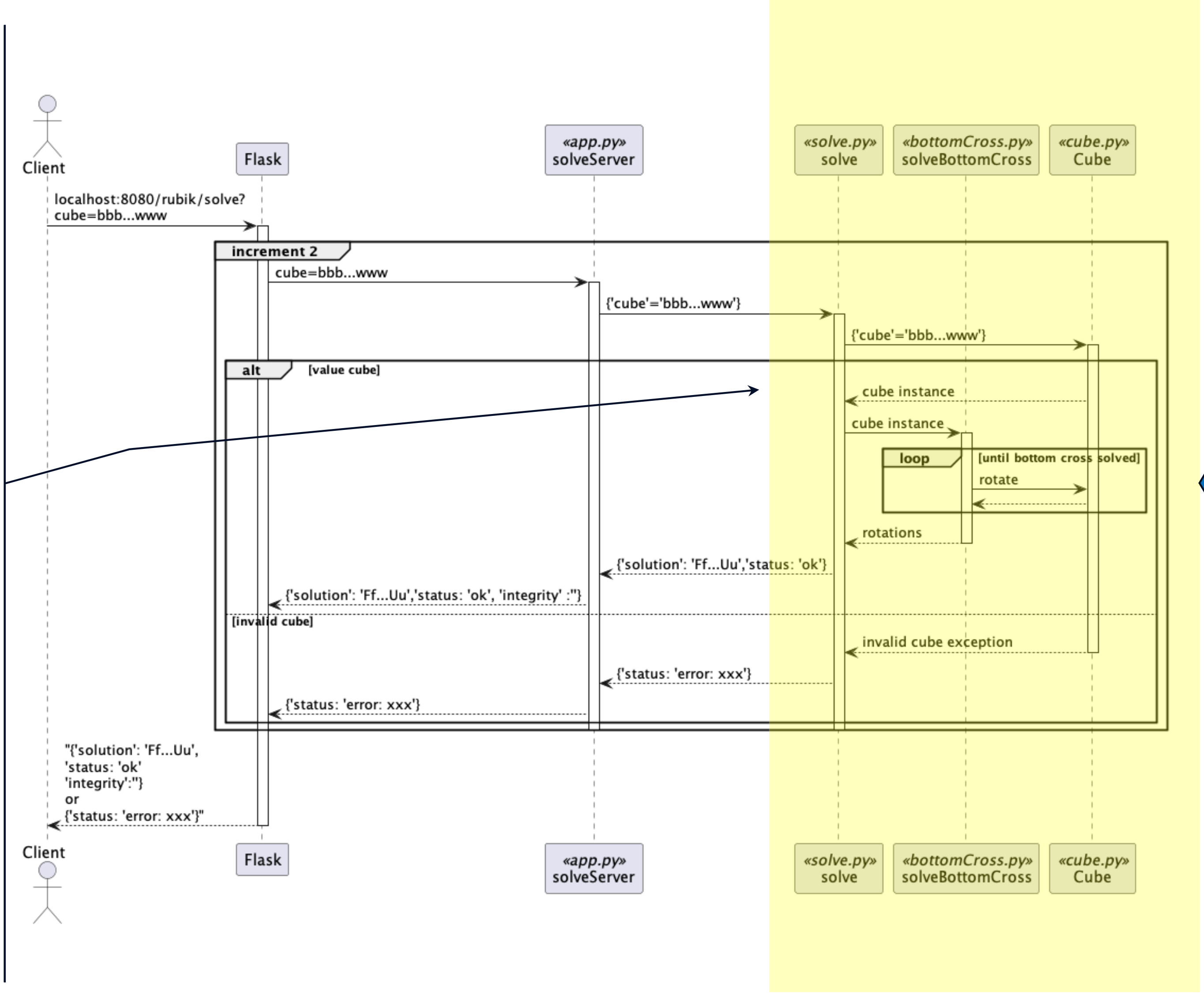
3. If the cube is value

3a.solve() passes the cube to solveBottomCross()

3b. solveBottomCross() performs rotations to achieve the down-face cross. It returns the series of rotations to solve()

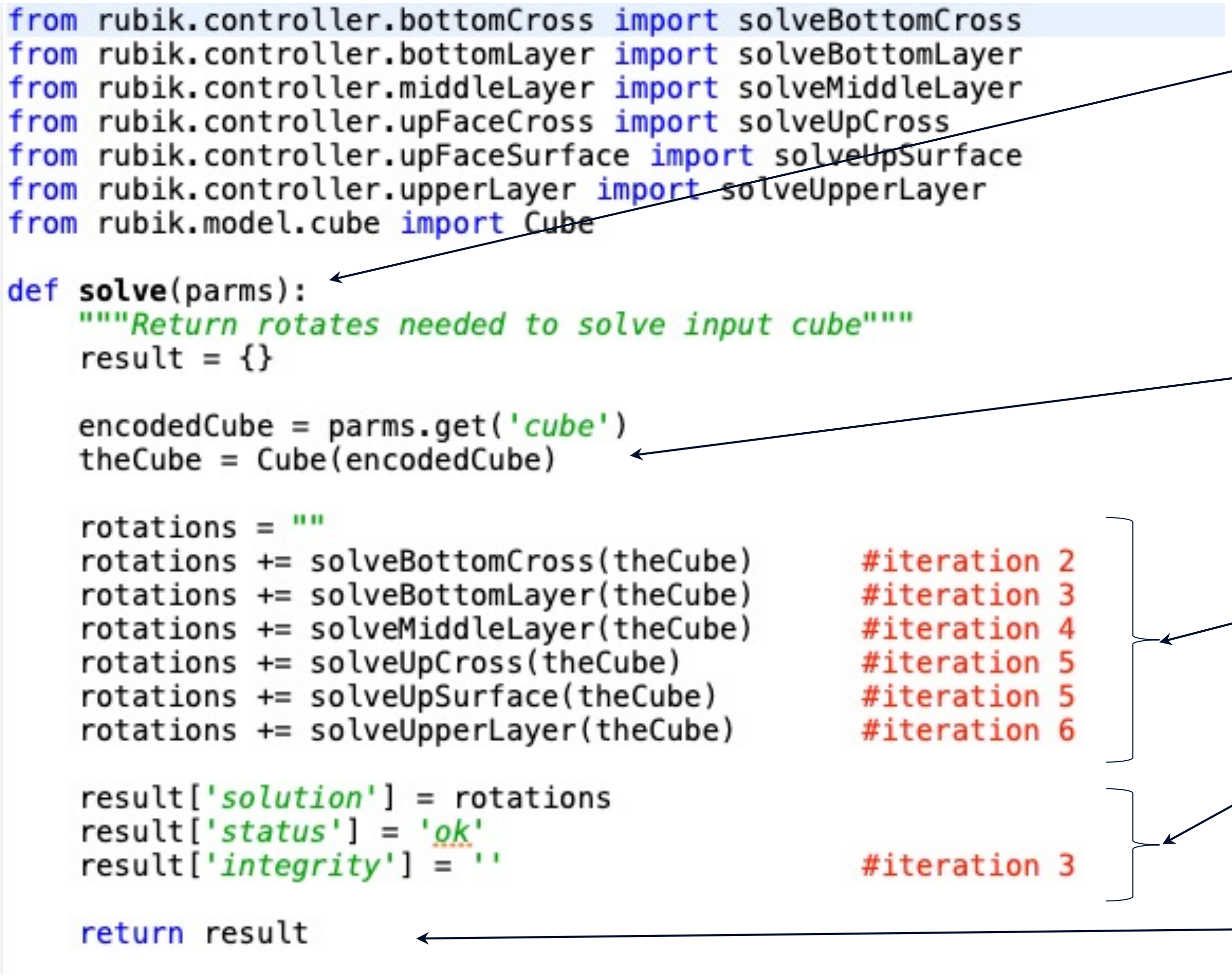
3c. solve() returns the rotations as the content of the "solution" key to solveServer()

4. if the cube is not valid, solve() returns an error status to solveServer()



This is your job for increment2

rubik/view/solve.py



```
from rubik.controller.bottomCross import solveBottomCross
from rubik.controller.bottomLayer import solveBottomLayer
from rubik.controller.middleLayer import solveMiddleLayer
from rubik.controller.upFaceCross import solveUpCross
from rubik.controller.upFaceSurface import solveUpSurface
from rubik.controller.upperLayer import solveUpperLayer
from rubik.model.cube import Cube

def solve(parms):
    """Return rotates needed to solve input cube"""
    result = {}

    encodedCube = parms.get('cube')
    theCube = Cube(encodedCube)

    rotations = ""
    rotations += solveBottomCross(theCube)      #iteration 2
    rotations += solveBottomLayer(theCube)      #iteration 3
    rotations += solveMiddleLayer(theCube)      #iteration 4
    rotations += solveUpCross(theCube)          #iteration 5
    rotations += solveUpSurface(theCube)        #iteration 5
    rotations += solveUpperLayer(theCube)       #iteration 6

    result['solution'] = rotations
    result['status'] = 'ok'
    result['integrity'] = ''                    #iteration 3

    return result
```

The query string portion of the URL is coming to you as a dictionary.

Keys and associated content are strings.

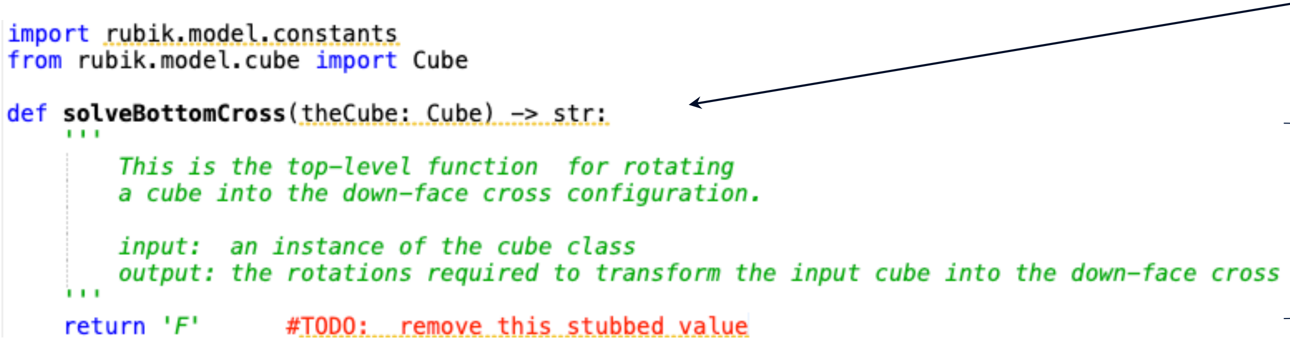
This instantiates the initial cube

This builds the solution in stages

This packages the information into a dictionary. (Note: 'integrity' will change in increment 3)

Return a dictionary. This is converted to a JSON string in solveServer(). IT IS NOT YOUR JOB TO DO THE CONVERSION.

rubik/controller/bottomCross.py



```
import rubik.model.constants
from rubik.model.cube import Cube

def solveBottomCross(theCube: Cube) -> str:
    '''
        This is the top-level function  for rotating
        a cube into the down-face cross configuration.

        input:  an instance of the cube class
        output: the rotations required to transform the input cube into the down-face cross
    '''
    return 'F'      #TODO:  remove this stubbed value
```

solveBottomCross() is given a cube instance

Your task is to determine the rotations

It returns a string representing the series of rotations that produce the down-face cross on the input cube.

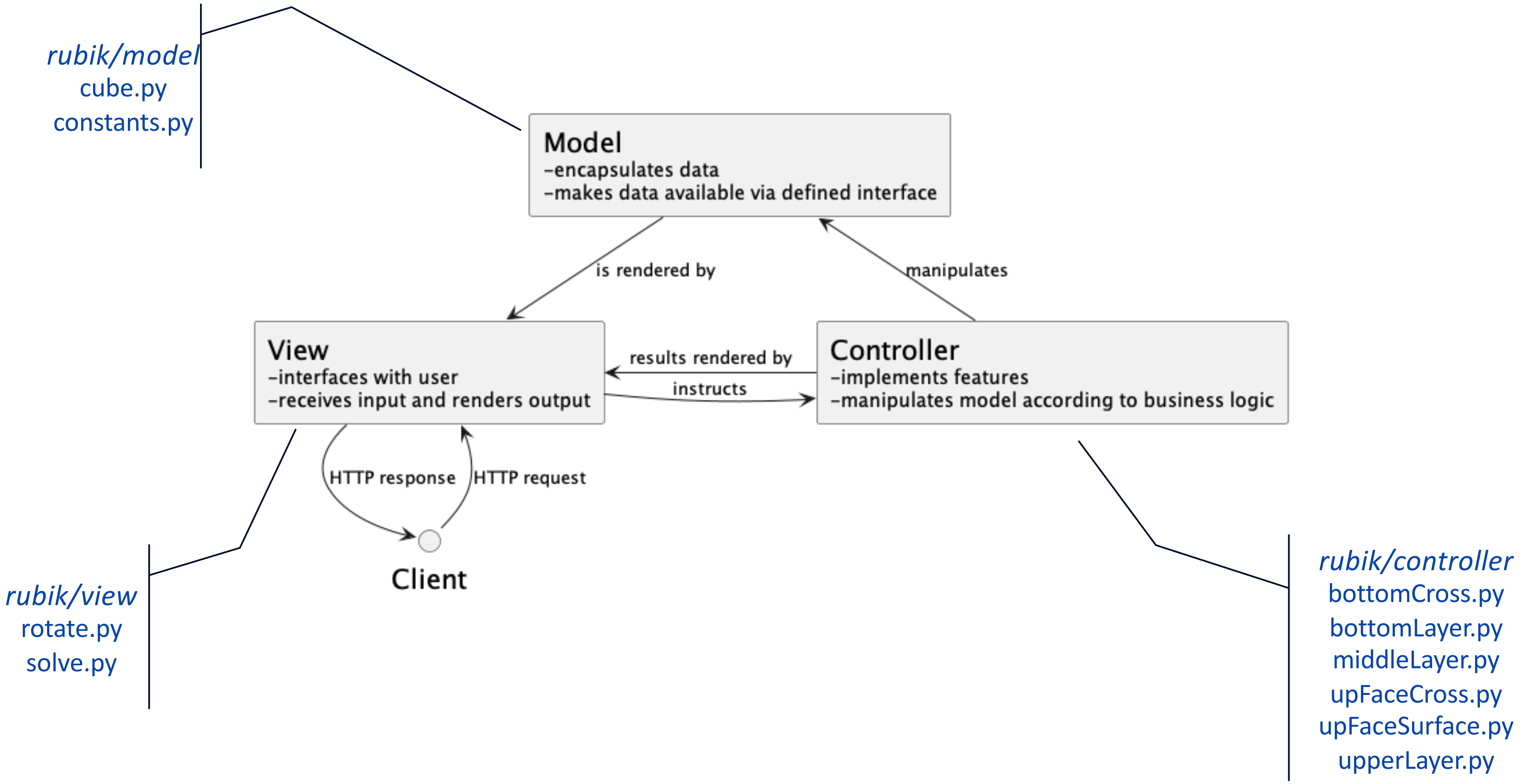
It currently returns a stubbed value

Add other methods/classes/scripts as you see fit.

Don't abandon good coding principles:

- write tests before production code
- readability over brevity
- readability over optimization
- sound cohesion
- loose coupling

Use TDD



rubik/model
cube.py
constants.py

Model
-encapsulates data
-makes data available via defined interface

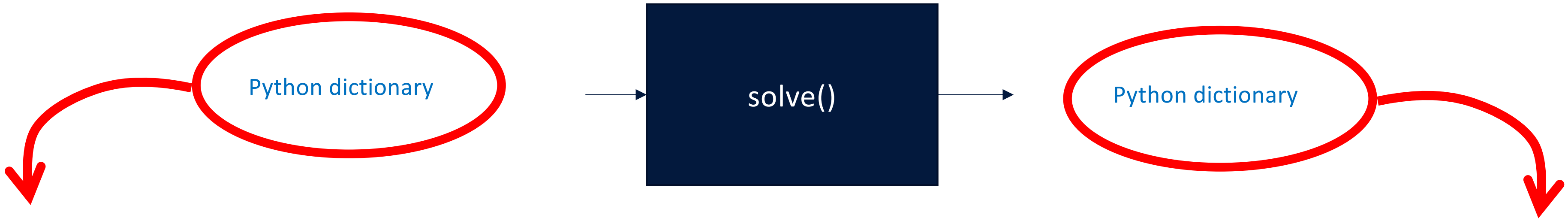
View
-interfaces with user
-receives input and renders output

Controller
-implements features
-manipulates model according to business logic

rubik/view
rotate.py
solve.py

rubik/controller
bottomCross.py
bottomLayer.py
middleLayer.py
upFaceCross.py
upFaceSurface.py
upperLayer.py

Product Aspect


Python dictionary

solve()

Python dictionary

Input representation of the cube

key:	‘cube’
value:	string - 54 characters - 6 unique characters from [a-zA-Z0-9] - 5th, 14th, 23rd, 32nd, 41st, and 50th characters must be unique
participation:	mandatory
understanding:	arrived unvalidated

No other keys are allowed

The output from solve() is a Python dictionary containing

If the input cube is valid :

{solution':'FfRrBbLlUu', 'status': 'ok', 'integrity':''}

where value of 'solution’ describes rotations

'status' is 'ok'

'integrity' is ''

If the input cube is invalid or extraneous keys are present:

{‘status’: ‘error: xxx’}

where ‘xxx’ is an error message of the developer’s choosing

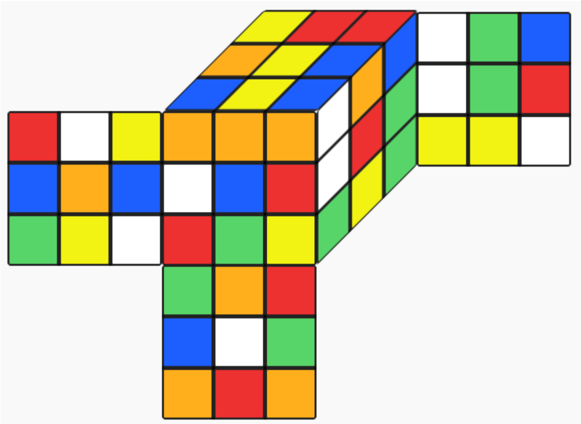
Product Aspect

URL

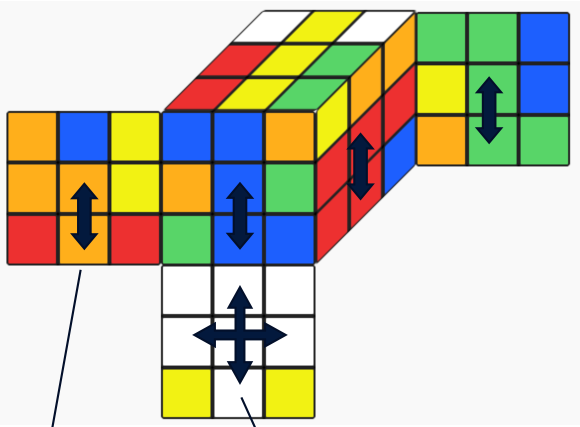
localhost:8080/rubik/solve?cube=ooowbrrgywobwrggygwgbwgryywrwybobgywyrroybbybgorbwgoro

Python dictionary passed to solve()

{'cube':'ooowbrrgywobwrggygwgbwgryywrwybobgywyrroybbybgorbwgoro'}



lfrbuFFRRBBLL



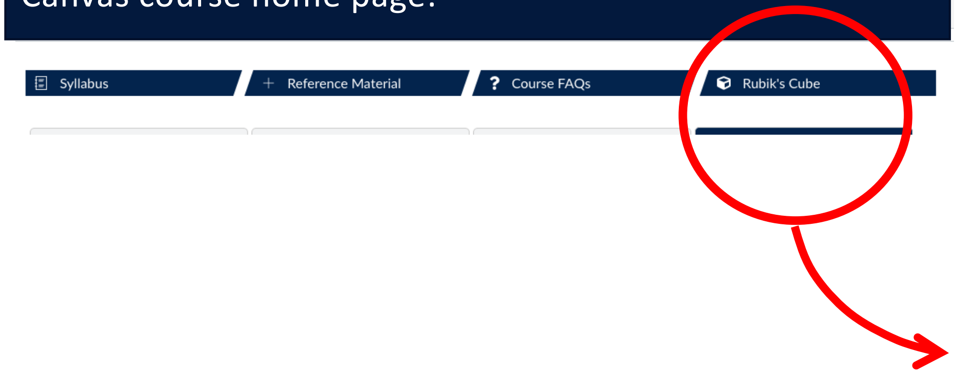
Python dictionary returned from solve()

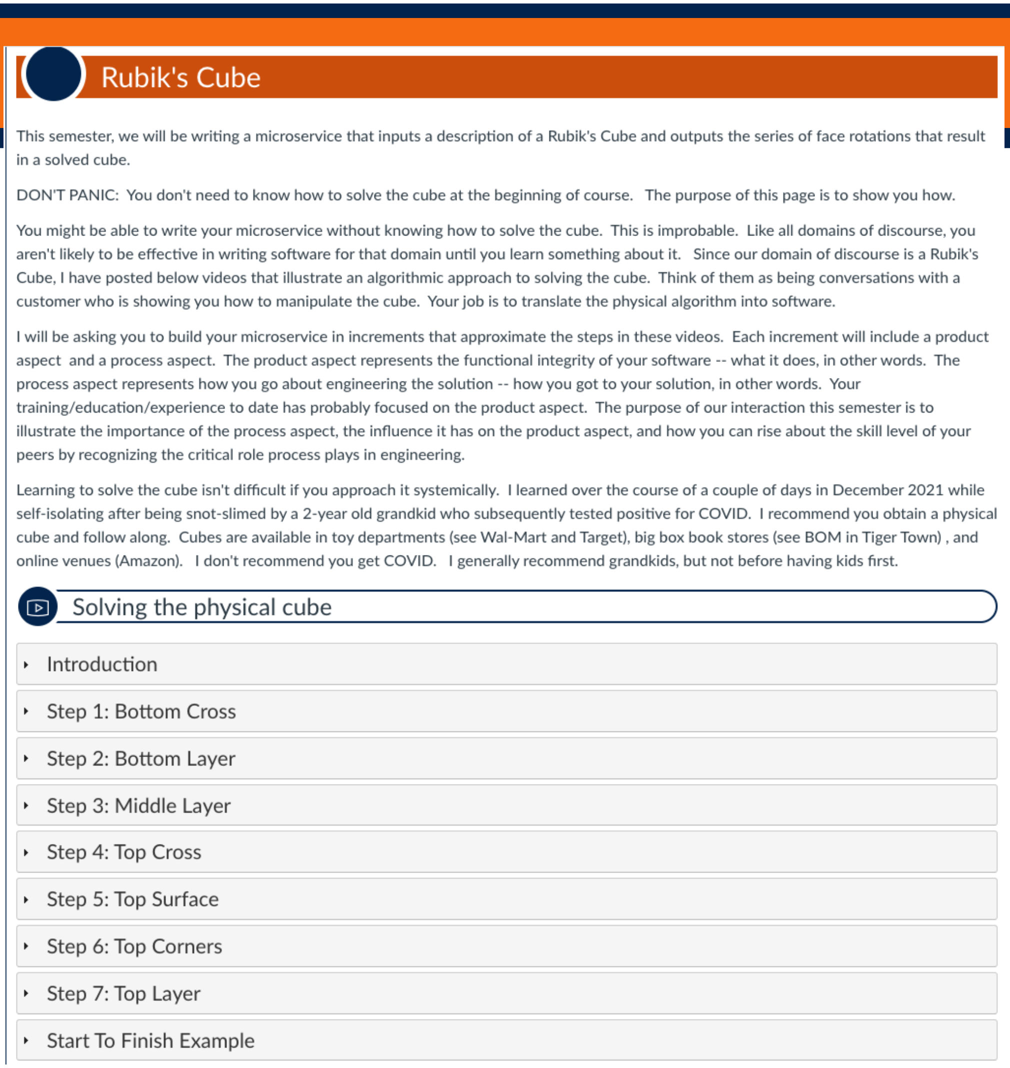
{'solution': ‘lfrbuFFRRBBLL', 'status': 'ok', 'integrity':''}

Bottom edge colors match middle color

Bottom cross

Canvas course home page:





Rubik's Cube

This semester, we will be writing a microservice that inputs a description of a Rubik's Cube and outputs the series of face rotations that result in a solved cube.

DON'T PANIC: You don't need to know how to solve the cube at the beginning of course. The purpose of this page is to show you how.

You might be able to write your microservice without knowing how to solve the cube. This is improbable. Like all domains of discourse, you aren't likely to be effective in writing software for that domain until you learn something about it. Since our domain of discourse is a Rubik's Cube, I have posted below videos that illustrate an algorithmic approach to solving the cube. Think of them as being conversations with a customer who is showing you how to manipulate the cube. Your job is to translate the physical algorithm into software.

I will be asking you to build your microservice in increments that approximate the steps in these videos. Each increment will include a product aspect and a process aspect. The product aspect represents the functional integrity of your software -- what it does, in other words. The process aspect represents how you go about engineering the solution -- how you got to your solution, in other words. Your training/education/experience to date has probably focused on the product aspect. The purpose of our interaction this semester is to illustrate the importance of the process aspect, the influence it has on the product aspect, and how you can rise about the skill level of your peers by recognizing the critical role process plays in engineering.

Learning to solve the cube isn't difficult if you approach it systemically. I learned over the course of a couple of days in December 2021 while self-isolating after being snot-slimed by a 2-year old grandkid who subsequently tested positive for COVID. I recommend you obtain a physical cube and follow along. Cubes are available in toy departments (see Wal-Mart and Target), big box book stores (see BOM in Tiger Town) , and online venues (Amazon). I don't recommend you get COVID. I generally recommend grandkids, but not before having kids first.

Solving the physical cube

- Introduction
- Step 1: Bottom Cross
- Step 2: Bottom Layer
- Step 3: Middle Layer
- Step 4: Top Cross
- Step 5: Top Surface
- Step 6: Top Corners
- Step 7: Top Layer
- Start To Finish Example

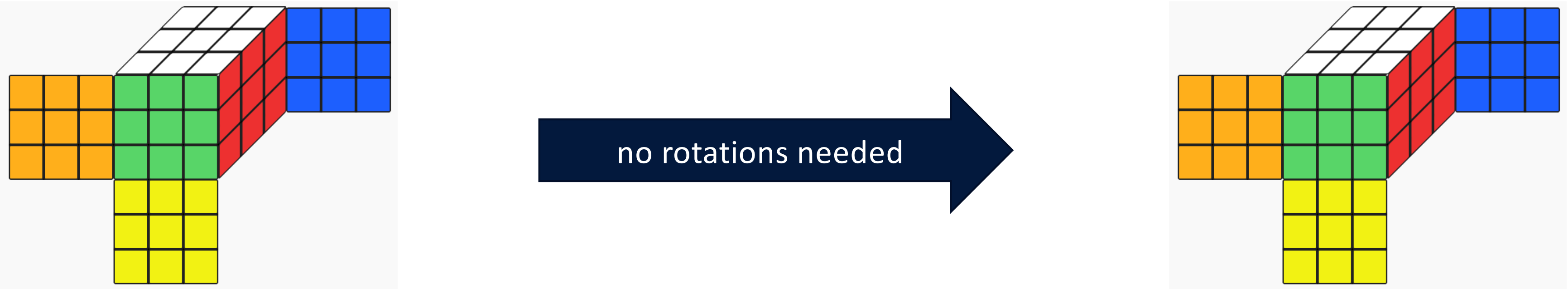
Product Aspect

URL

localhost:8080/rubik/solve?cube=gggggggggrrrrrrrrrbbbbbbbbboooooooooowwwwwwwwwyyyyyyyyy

Python dictionary passed to solve()

{'cube':'gggggggggrrrrrrrrrbbbbbbbbboooooooooowwwwwwwwwyyyyyyyyy'}

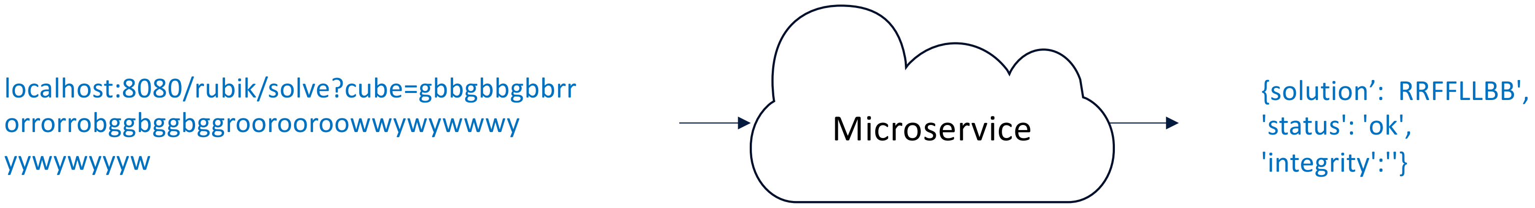


no rotations needed

Python dictionary returned from solve()

{'solution': '', 'status': 'ok', 'integrity':''}

Process Aspect



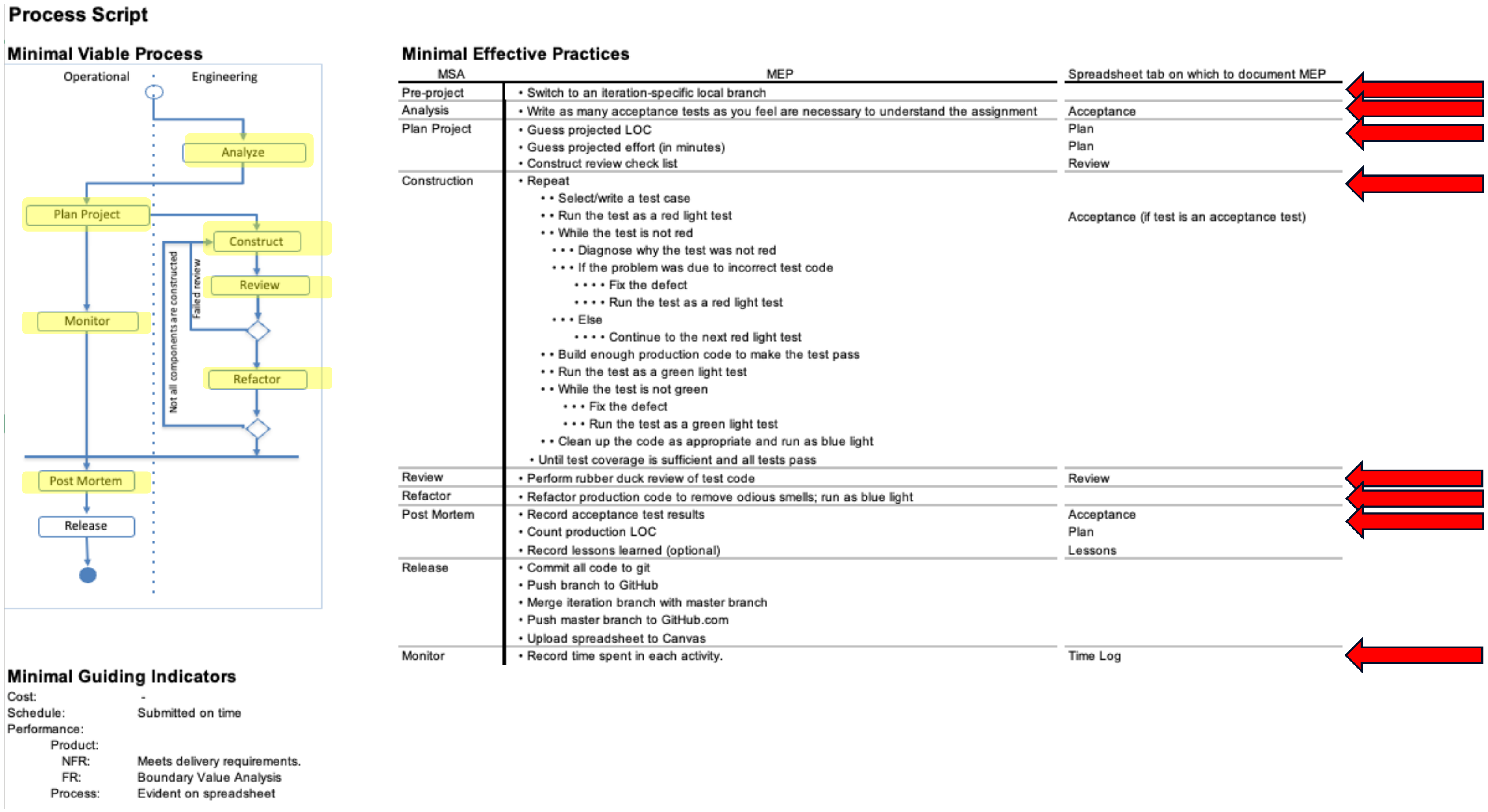
localhost:8080/rubik/solve?cube=gbbgbbgbbrr
orrorrobggbggbggroorooroowwywywwwy
yywywyyyw

{solution’: RRFFLLBB',
'status': 'ok',
'integrity':''}

Process objectives:

- to have you hone your TDD skills
- to have you conduct a rubber duck review
- to have you eliminate code smells

Process Aspect



Process Script

Minimal Viable Process

Minimal Effective Practices

MSA	MEP	Spreadsheet tab on which to document MEP
Pre-project	• Switch to an iteration-specific local branch	
Analysis	• Write as many acceptance tests as you feel are necessary to understand the assignment	Acceptance
Plan Project	• Guess projected LOC	Plan
	• Guess projected effort (in minutes)	Plan
	• Construct review check list	Review
Construction	• Repeat	
	• • Select/write a test case	
	• • Run the test as a red light test	Acceptance (if test is an acceptance test)
	• • While the test is not red	
	• • • Diagnose why the test was not red	
	• • • If the problem was due to incorrect test code	
	• • • • Fix the defect	
	• • • • Run the test as a red light test	
	• • • Else	
	• • • • Continue to the next red light test	
	• • Build enough production code to make the test pass	
	• • Run the test as a green light test	
	• • While the test is not green	
	• • • Fix the defect	
	• • • Run the test as a green light test	
	• • Clean up the code as appropriate and run as blue light	
	• Until test coverage is sufficient and all tests pass	
Review	• Perform rubber duck review of test code	Review
Refactor	• Refactor production code to remove odious smells; run as blue light	
Post Mortem	• Record acceptance test results	Acceptance
	• Count production LOC	Plan
	• Record lessons learned (optional)	Lessons
Release	• Commit all code to git	
	• Push branch to GitHub	
	• Merge iteration branch with master branch	
	• Push master branch to GitHub.com	
	• Upload spreadsheet to Canvas	
Monitor	• Record time spent in each activity.	Time Log

Minimal Guiding Indicators

Cost:		-
Schedule:		Submitted on time
Performance:		
	Product:	
	NFR:	Meets delivery requirements.
	FR:	Boundary Value Analysis
	Process:	Evident on spreadsheet

Smells

- Please refactor these "odious smells"
  - duplicate code
    - example: repeated code segments (either exact or similar repetition)
  - long methods
    - threshold: 25 LOC
  - long classes
    - threshold: 25 public methods
  - temporary variables
    - examples: temp, tmp, x, i, etc.
  - useless comments
    - example: comments that are obvious, comments that explain unreadable code
  - private parts
    - example: referencing an attribute of a class without using a getter/setter
  - coding standard violations
    - example: magic numbers
    - example: variable names that are not meaningful

Process Aspect

Deliverables

- code pushed to GitHub
- spreadsheet submitted to Canvas