Functions, File I/O and Strings

Jin-Soo Kim
(jinsoo.kim@snu.ac.kr)

Systems Software & Architecture Lab.

Seoul National University

Dec 16 – 20, 2019

Index

- Testing (30’)
  - Summation
- Basic Lab (45’) × 3
  - 색상 코드 2
  - 하노이의 탑 2
  - Caesar Cipher
- Advanced Lab (75’)
  - Differentiation

Testing

Test Driven Development

- 기존의 방식은 코드를 짠 다음에 테스트를 진행하는 것
- **코드를 짜기 전에** 테스트를 진행하는 것이 TDD의 핵심
  - Test Case(입력 & 예상 출력)를 작성하고 이를 검증하는 코딩 수행

- 장점
  - 테스트 케이스를 작성하면서 본인이 구현하려는 로직을 더 깊이 이해할 수 있음
  - 코드를 수정할 때 부담감을 줄일 수 있음

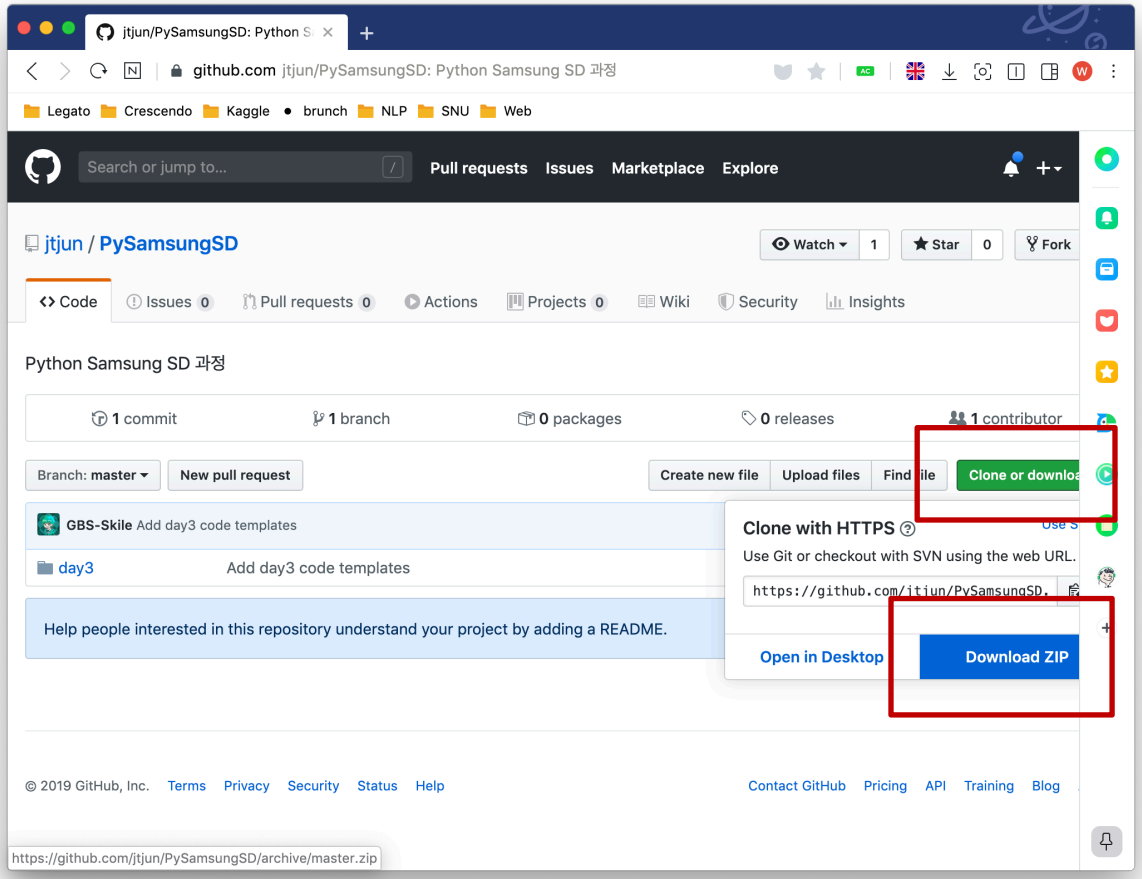
- 단점
  - 테스트 케이스 작성 과정이 번거로울 수 있음

Making Test Cases

- 테스트 케이스를 잘 만드는 방법
  - 일단, 많을 수록 좋다
  - 랜덤 값을 이용하여 다수의 테스트 케이스 만들기
  - 오류가 자주 발생하는 특수한 상황에 대한 테스트 케이스 만들기
    - 허용되는 최솟값, 최댓값에 대하여
    - 극값 주변의 값도 가급적 함께 검사
    - 비어 있는 데이터 (0, None, empty string)에 대하여 검사
    - 그 외 문제 상황에서 고려할 수 있는 이상한 값들 검사

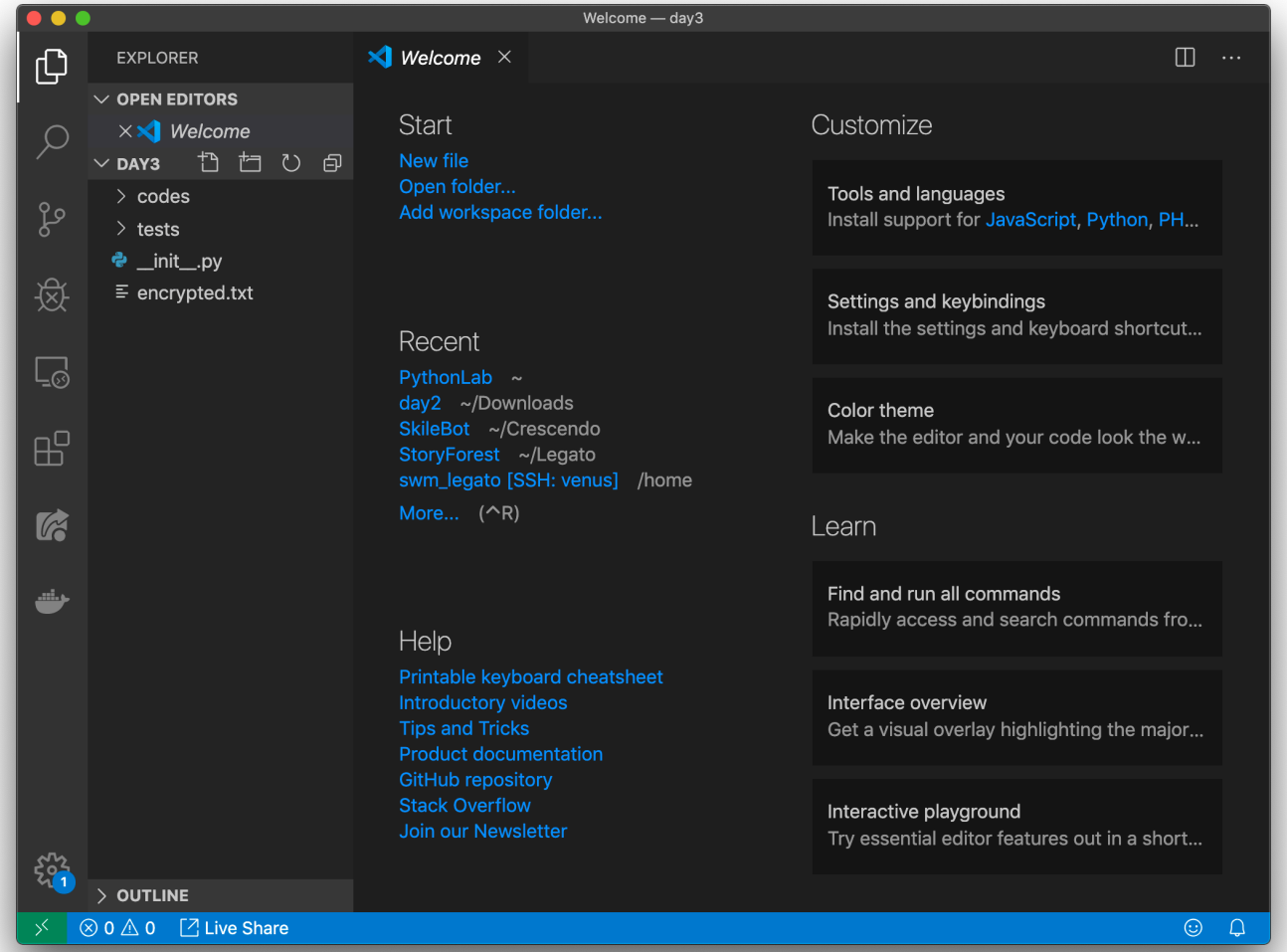
코드 템플릿

- https://github.com/jtjun/PySamsungSD



코드 템플릿

- VSCode – File – Open… - '*day3*' 폴더 선택



Example: Summation Function

- "합"을 구하는 프로그램 만들기
  - 다음 세 조건식을 모두 성립시키는 함수를 생각해보기
    - `assert [조건식]` : [조건식]이 False인 경우 오류 강제로 발생

```
# Case #1: two keyword argument
assert my_sum(a=1, b=2) == 3
assert my_sum(a=0, b=300) == 300
assert my_sum(a=-200, b=0) == -200
```

Example: Summation Function

- "합"을 구하는 프로그램 만들기
  - 인수가 하나인 경우 **추가**, 기존 Test Case #1을 유지해야 함

```
# Case #2: one argument
assert my_sum('text') == 'text'
assert my_sum(2 ** 100) == 2 ** 100
assert my_sum(a=-3.21e4) == -3.21e4
```

  - 매개변수 b는 값이 있을 수도 없을 수도 있다 → **디폴트 매개변수**

Example: Summation Function

- "합"을 구하는 프로그램 만들기
  - 인수가 여러 개일 때 **추가**, 기존 Test Case #1, #2 유지

```
# Case #3: multiple arguments
assert my_sum(1, 2, 3) == 6
assert my_sum(1, 2, 3, 4, 5, -1, -2, -3, -4, -5) == 0
assert my_sum(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1) == 1
```

  - 매개변수의 개수가 고정되지 않았다 → **가변 매개변수**(**args*) 활용

Example: Summation Function

- "합"을 구하는 프로그램 만들기
  - 인수가 없을 때 **추가**, 기존 Test Case #1, #2, #3 유지

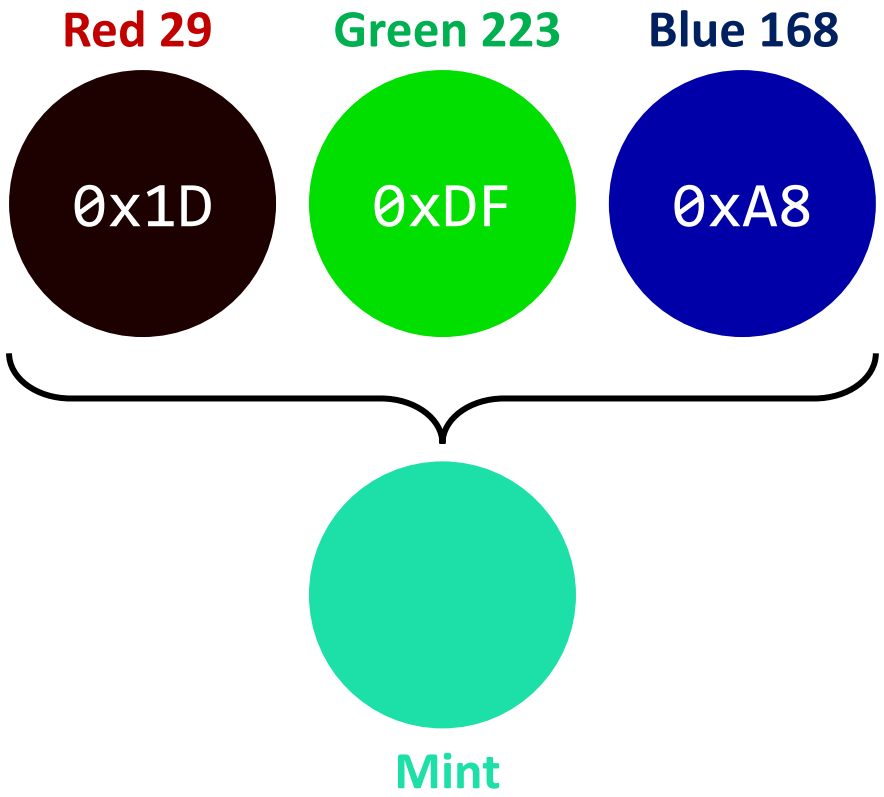
```
# Case #4: without any parameters
try:
    my_sum()
    exit(1)             # exit if no error occurred
except Exception:       # catch every exception
    pass                # ...and do nothing

print("All test case passed!")
```

Basic Lab

Lab 3-1. 색상 코드 2

- Lab 1-6 "색상 코드"를 기억하시나요?
  - 16진수로 붙여 표현하는 방법 #RRGGBB (ex: **#1DDFA8**)
  - 10진수로 표현하는 방법 rgb(RRR, GGG, BBB) (ex: **rgb(29,223,168)**)



Lab 3-1. 색상 코드 2

- `def convert(color_code):`
  - *color_code*: '16진수 표기법' 또는 '10진수 표기법'에 해당하는 문자열
    - 16진수 표기법 : "#***RRGGBB***" 형식이며 R, G, B는 16진수 한 글자
    - 10진수 표기법 : "`rgb(`***R***`,`***G***`,`***B***`)`" 형식이며 R, G, B는 10진수 0~255

  - Expected outputs
    - color_code가 16진수 표기법이었을 경우, 동일한 색상에 대한 10진수 표기법 출력
    - color_code가 10진수 표기법이었을 경우, 동일한 색상에 대한 16진수 표기법 출력

```
assert convert('#c0ffee') == 'rgb(192,255,238)'
assert convert('rgb(192,255,238)') == '#c0ffee'
```

Lab 3-1. 색상 코드 2

- Hints
  - 고정된 길이의 부분문자열을 구할 때는 String Slicing 사용하기 (Day 1)
    - s번째부터 e번째까지의 부분문자열을 구하고 싶다면: `string[s : e+1]`
  - 16진수 string을 int로 변환할 때는 `int('', 16)` 사용하기 (Day 1)
  - 주어진 입력의 형식 판별할 때 조건문(if) 사용하기 (Day 2)
  - 특정 토큰(예를들어, comma)을 기준으로 문자열 쪼갤 때 `string.split('token')` 사용하기 (Day 3)
  - int를 16진수 string으로 변환하고 싶을 때 `"%x" % i` 사용하기 (Day 3)

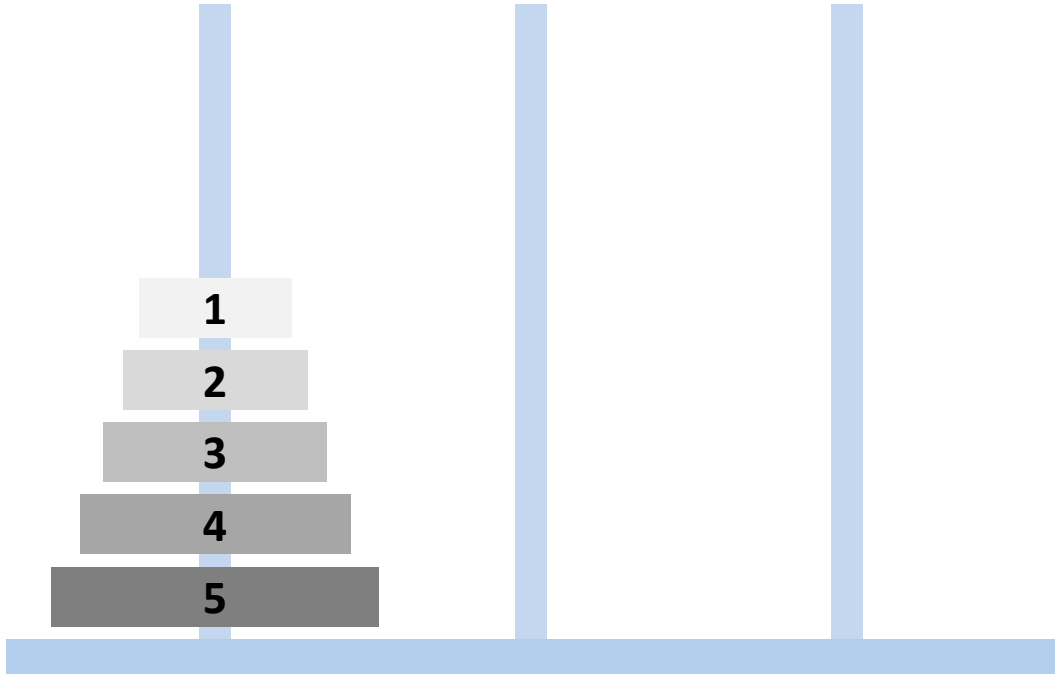
Lab 3-2. 하노이의 탑 2

- 수학적 귀납법의 아이디어를 이용하여 재귀적으로 문제 해결하기
  - **조건 1** : N=1일 때, 문제는 자명하게 해결된다.
  - **조건 2**
    - N=k일 때 문제를 해결하는 방법을 알고 있다고 가정
    - 해당 가정 밑에서 N=k+1일 때도 문제를 해결할 수 있음을 증명

  - **조건 1과 조건 2를 만족하면, 모든 자연수 N에 대해 문제를 해결할 수 있다!**

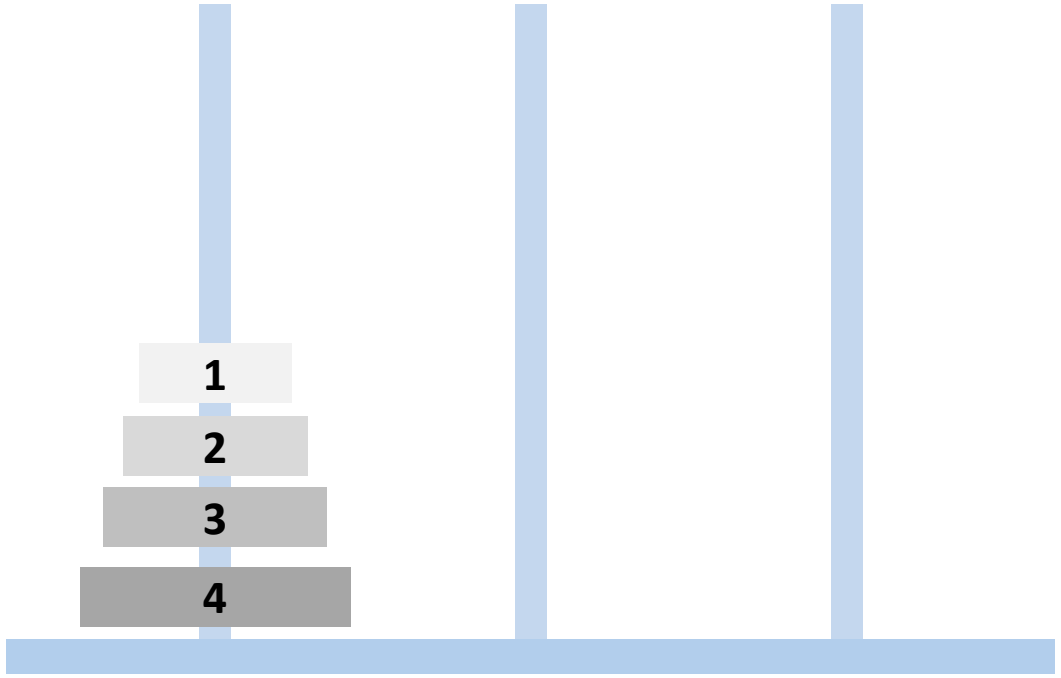
Lab 3-2. 하노이의 탑 2

- 예시 : N = 5 하노이 탑 문제
  - N = 4 하노이 탑 문제를 해결할 수 있다고 가정



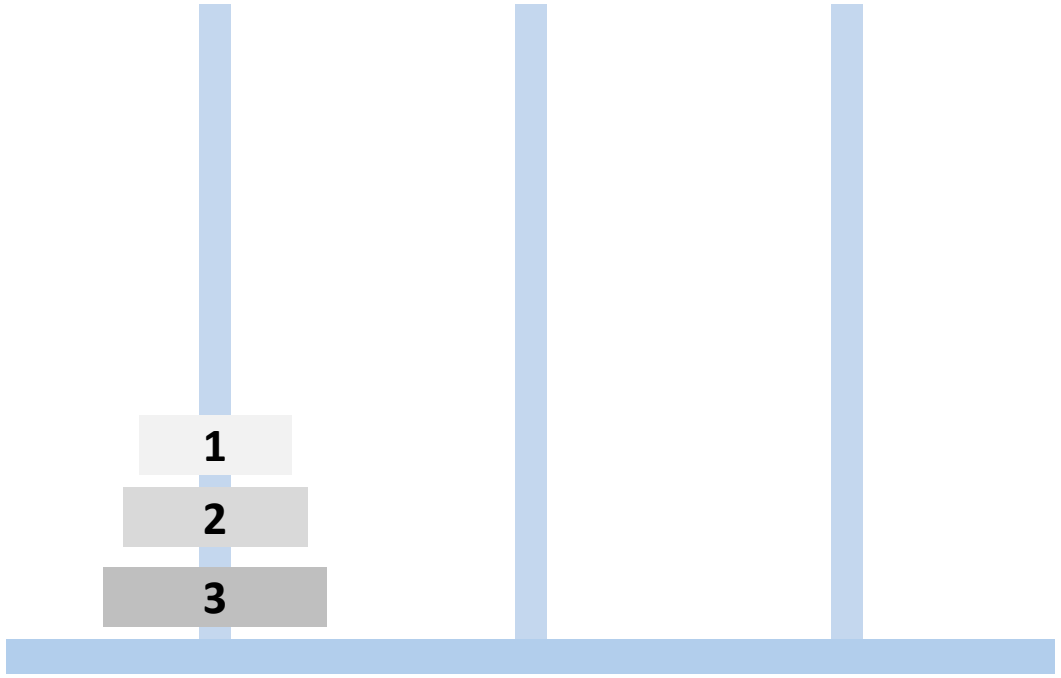
Lab 3-2. 하노이의 탑 2

- 예시 : N = 5 하노이 탑 문제
  - 그럼, N = 4 하노이 탑 문제는 어떻게 풀 건데?
    - N = 3일 때 문제를 해결할 수 있다고 가정~



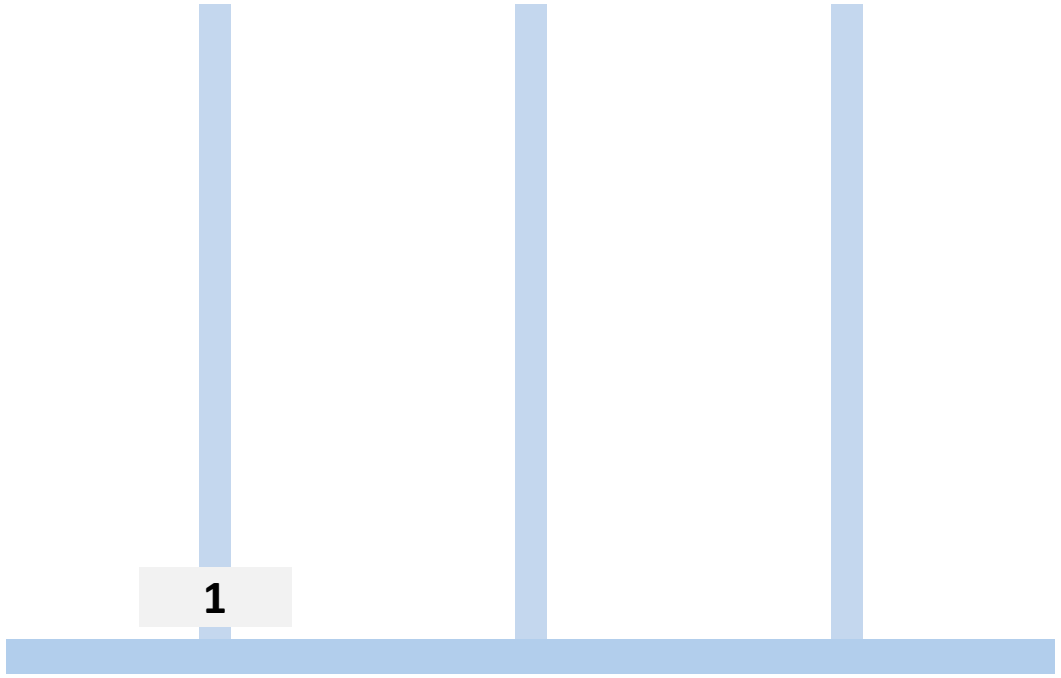
Lab 3-2. 하노이의 탑 2

- 예시 : N = 5 하노이 탑 문제
  - … 그럼, N = 3은?
    - N = 2일 때!



Lab 3-2. 하노이의 탑 2

- 예시 : N = 5 하노이 탑 문제
  - 이 논리는 N=1일 때는 적용할 수 없다 (N > 0)
  - 하지만 자명하게 해결할 수 있다



Lab 3-2. 하노이의 탑 2

- `def play_hanoi(n):`
  - *n*: 원판의 개수
  - Expected outputs
    - 세 개의 기둥을 각각 0, 1, 2로 표현할 때,
    - 0에서 1로 n개의 원판 전체를 옮기는 과정을 기술한다.
    - 각각의 움직임(move)은 길이 2짜리 리스트 `[pole_from, pole_to]` 로 기술
    - 움직임들의 리스트로 정답 기술

```
assert play_hanoi(1) == [[0, 1]]
assert play_hanoi(2) == [[0, 2], [0, 1], [2, 1]]
```

Lab 3-2. 하노이의 탑 2

- 재귀적 접근 방법
  - (Base case) `n == 1`, return What?
  - (Induction) `play_hanoi(n-1)`로 `play_hanoi(n)` 만들기
    - **#1: (n-1)개 원판을 0에서 2로 옮긴다**
    - #2: 가장 큰 원판을 0에서 1로 옮긴다
    - **#3: #1에서 옮겼던 (n-1)개 원판을 2에서 1로 옮긴다**
    - return [#1] + [#2] + [#3]

  - [#1], [#3]은 `play_hanoi(n-1)`만으로는 해결할 수 없어 보인다
    - 판의 초기 위치와 목표 위치 정보가 추가로 주어져야 한다

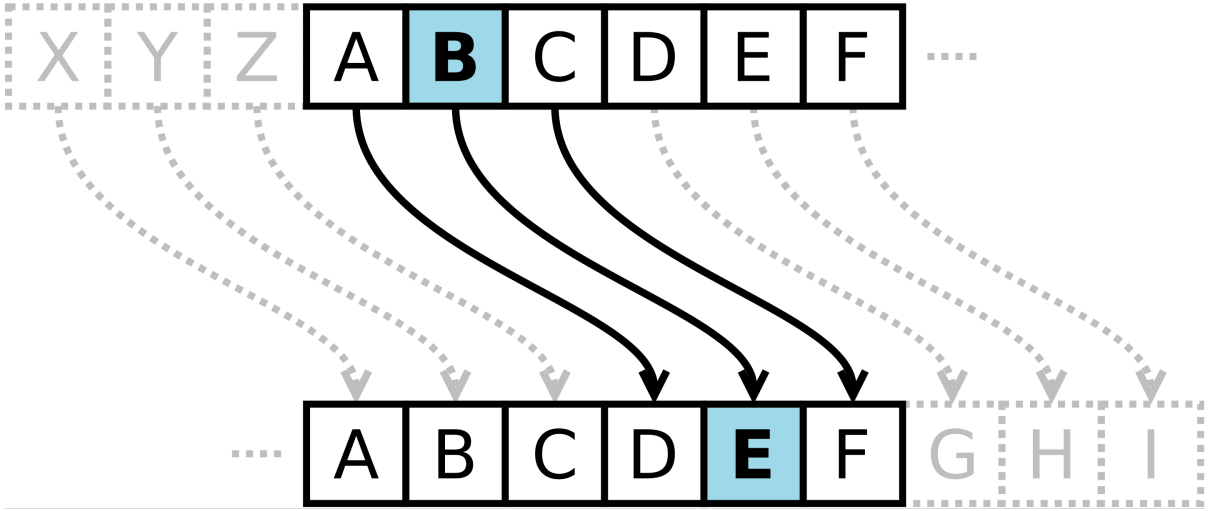
Lab 3-2. 하노이의 탑 2

- *re*-`def play_hanoi(n, p_from=0, p_to=1):`
  - (Base case) `n == 1`, return **What?**
  - (Induction) `play_hanoi(n-1)`로 `play_hanoi(n)` 만들기
    - #1: (n-1)개 원판을 ***p_from***에서 ***[나머지 기둥]***로 옮긴다
    - #2: 가장 큰 원판을 ***p_from***에서 ***p_to***로 옮긴다
    - #3: #1에서 옮겼던 (n-1)개 원판을 ***[나머지 기둥]***에서 ***p_to***로 옮긴다
    - return [#1] + [#2] + [#3]

  - ***[나머지 기둥]***은 어떻게 계산하는가?

Lab 3-3. Caesar Cipher

- **카이사르 암호** : 알파벳에 적용할 수 있는 암호화 규칙
  - 열쇠값(key)을 정하고, 각 알파벳을 열쇠값만큼 뒤로 밀어 암호화
  - 열쇠값이 3인 경우 (ZEBRA → CHEUD)



  - 열쇠값만큼 알파벳을 앞으로 당기면, 복호화도 가능함

Lab 3-3. Caesar Cipher

- 실습 파일에 들어있는 *encrypted.txt* 해독하기!

encrypted.txt

```
2 Vjg Bgp qh Ravjqp, da Vko Rgvgtu

8 Jmicbqnct qa jmbbmz bpiv cotg.
15 Tmeaxrxi xh qtiitg iwpc xbeaxrxi.
…
```

- 아마 각 줄의 첫 번째 수가 key인 듯 보인다…

Lab 3-3. Caesar Cipher

- `def encrypt(text, key):`
  - *text*
    - 암호화할 문자열
  - *key*
    - 암호화에 사용할 정수
  - Expected Output
    - 암호화된 문자열
    - 대문자 알파벳끼리 key만큼 밀어내기
    - 소문자 알파벳끼리 key만큼 밀어내기
    - 이외의 문자는 그대로 출력하기

- `def decrypt(text, key):`

```
1   BIG = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
2   SMALL = "abcdefghijklmnopqrstuvwxyz"
3
4   def encrypt(text, key):
5       ????????
6       ????????
7       ????????
8       ...
9       ????????
10      return ????
11
12  def decrypt(text, key):
13      return encrypt(????, ????)
```

Lab 3-3. Caesar Cipher

- Hints
  - 문자열을 구성하는 각각의 문자에 대해 반복문 돌리기 (`for char in text:`)
  - 정의한 BIG, SMALL 상수를 활용하기
    - char이 알파벳 대문자인가? 그렇다면, 몇 번째 알파벳인가?
    - char이 알파벳 소문자인가? 그렇다면, 몇 번째 알파벳인가?
    - `in` 연산자 활용 & `string.index()` (또는 `string.find()`) 활용
  - 알파벳 위치를 알았다면, key만큼 밀어내기 (Day 1 모듈러 연산)
  - key만큼 밀어냈다면, 해당 위치 알파벳 구하기 (String Indexing)
  - 문자열을 더해가며 암호문 완성하기 (`result += char`)

Lab 3-3. Caesar Cipher

- 파일 입출력 활용하여 문제의 *encrypted.txt* 해독하기
  - 공백"만" 있는 줄을 처리하는 방법
    - `string.strip()`으로 양 옆 공백을 제거하거나,
    - 줄에 숫자가 있는지 확인하거나,
    - …
  - key와 암호문을 분리하는 방법
    - key의 범위가 1~25라는 점을 활용하여 앞 두 글자만을 취하거나,
    - 해당 문장의 첫 번째 space 위치를 `string.find()`로 찾거나,
    - …
  - 앞서 만든 `decrypt(text, key)` 활용하여 평문 출력하기

Advanced Lab

Lab 3-4. Differentiation

- 최종적으로 만들고자 하는 그림
  - We can do it!!!

```
assert d_dx("5x^2 + 2x + -7") == "10x + 2"
assert d_dx("x^4 + -7x") == "4x^3 + -7"
assert d_dx("1 + x + 2x + 3x^2") == "1 + 2 + 6x"
```

Lab 3-4. Differentiation

- 구체적인 제약조건
  - **다항함수 문자열**을 입력받았을 때, 그 미분함수를 **다항함수 문자열**로 출력하기

  - **다항함수 문자열**
    - 다항함수 문자열은 최소 1개 이상의 항으로 구성되어 있어야 한다.
    - 각각의 항은 덧셈(`'+'`) 기호로 구분되어 있으며, 덧셈 기호 사이에는 한 칸의 공백이 있다.
      - `'[first_term] + [second_term] + … + [last_term]'`

    - 항은 (정수) **계수**와 (음이 아닌 정수) **차수**로 표현할 수 있다.
      - 차수가 0인 경우, 이 항을 **상수항**이라 하며, 정수 계수를 있는 그대로 출력한다. (예: `'-1'`, `'0'`, `'1'`)
      - 차수가 1 이상이면, 정수 계수 뒤에 문자 x를 붙인다.
        추가로, 차수가 2 이상이면, 문자 x 뒤에 기호 ^와 차수를 붙인다. (예: `'-5x^2'`, `'0x'`, `'3x^4'`)
      - 차수가 1이상이고 계수의 절댓값이 1인 경우, 숫자 1을 생략한다. (예: `'-x'`, `'x^2'`, `'-x^3'`)

Lab 3-4. Differentiation

- 구체적인 제약조건
  - 미분함수
    - 어느 임의의 다항함수의 미분함수는 여러 **다항함수 문자열**로 표현 가능하다.

  - 예 : equation = `'x^2 + 2x + 1'`
    - `'2x + 2 + 0'`
    - `'2x + 2'`
    - `'2 + 2x'`
    - `'1 + x + 1 + x + 0x^2 + 0x^3'`

  - 채점 코드는 항의 순서, 계수가 0인 항, 동류항 묶기 등등을 신경쓰지 않는다!
  - 신경 쓸만한 부분은, 다항함수 문자열은 **최소 1개의 항**이 필요하다는 점
    - 빈 문자열 `''`을 출력하면 안 된다. 최소한 `'0'`은 출력해야 함

Lab 3-4. Differentiation

- `def print_term(factor, degree):`
  - *factor* : 항의 계수 (정수)
    - 음수인 경우?
    - 절댓값이 1인 경우?
  - *degree* : 항의 차수 (0 이상의 정수)
    - 0인 경우?
    - 1인 경우?

```
assert print_term(0, 2) == "0x^2"
assert print_term(-1, 1) == "-x"
assert print_term(5, 0) == "5"
```

Lab 3-4. Differentiation

- `def print_equation(terms):`
  - *terms* : list
    - 각 원소는 [factor : int, degree : int]의 형태
  - Expected output
    - `' + '` 문자를 기준으로 합침 : `string.join()`?

```
assert print_equation(
  [[0, 2], [-1, 1], [5, 0]]
) == "0x^2 + -x + 5"
```

Lab 3-4. Differentiation

- `def parse_term(term_str):`
  - `print_term()`의 역함수 꼴
  - *term_str* : 항을 문자열로 표현

  - Expected outputs
    - list : [factor : int, degree : int]

```
assert parse_term("0x^2") == [0, 2]
assert parse_term("-x") == [-1, 1]
assert parse_term("5") == [5, 0]
```

Lab 3-4. Differentiation

- `def parse_equation(equation):`
  - *equation* : str
    - `' + '` 문자를 기준으로 쪼개기 : `string.split()`?
  - Expected output

```
assert parse_equation("0x^2 + -x + 5") == \
    [[0, 2], [-1, 1], [5, 0]]
```

Lab 3-4. Differentiation

- `def d_dx_as_terms(terms):`
  - *terms* : list of list
    - [[1항 계수, 1항 차수], [2항 계수, 2항 차수], …, [3항 계수, 3항 차수]]
  - Expected output
    - 형식은 *terms*와 동일
    - 가능한 답은 여러가지가 될 수 있음!
    - 신경 쓸 부분 (**다항함수 조건**)
      - 출력된 계수는 정수인가
      - 출력된 차수는 0 이상의 정수인가
      - 항의 개수는 1 이상인가
      - 미분 법칙에 잘 맞는가 : $(\frac{d}{dx}ax^n = (an)x^{n-1}, \frac{d}{dx}a = 0$ and $\frac{d}{dx}(f+g) = \frac{d}{dx}f + \frac{d}{dx}g)$

Lab 3-4. Differentiation

- `def d_dx(equation):`
  - *equation* : str
  - Expected output : str

  - 지금껏 구현했던 함수들을 총동원하기
  - 3줄로 간결하게 표현할 수 있음

Submission

Lab 3 Submission

- `lab_3_4.py` 소스 코드 제출 (성함과 함께)
  - tothesky7@snu.ac.kr

  - 할 수 있는 만큼만 열심히 코드를 작성하셔서 보내주시면 됩니다.
    - 모든 테스트 케이스를 다 맞추시지 않아도 괜찮습니다.
  - 템플릿 하단의 테스트 코드는 제외하셔도 됩니다.
  - 12월 19일(목)까지 받습니다.

  - 따로 요청이 없으면 별도의 피드백(답장)은 보내드리지 않습니다.