생명과학 융합 프로그래밍 2

2019년 7월 13일

황 승 민

생명과학 융합 프로그래밍 2 목차

- 참고 : 프로그래밍 기법 연습
- 도입 : 멘델의 유전법칙 그리고 현대적 해석
- 문제 1 : 단일 대립 유전자 유전 현상 해석
- 문제 2 : 여러 대립 유전자 유전 현상 해석 (독립인 경우)
- 문제 3 : OOP를 이용한 염색체 모델링 (심화)

프로그래밍 기법 연습

프로그래밍 기법 연습

- 이번 수업 때 사용할 자료 구조 및 프로그래밍 기법 (Python 사용)
  - 문자열 처리
  - 리스트 (리스트 순회, 리스트 슬라이싱 등)
  - 재귀함수
  - 객체지향프로그래밍 (class)

프로그래밍 기법 - 문자열

```
>>> str1 = "string"
>>> str2 = "practice"
>>> str1[1]
't'
>>> str1 + " " + str2
'string practice'
>>> str1[1:3]
‘tr'
>>> str1[:3]
'str'
```

프로그래밍 기법 - 재귀함수

For 문을 재귀함수로 나타내기

```
(for)
sum = 0
for i in range(10):
    sum += i

(recursion)
sum = 0
def loop(i, n):
    global sum
    if(i >= n):
        return sum
    else:
        sum += i
        loop(i + 1, n)
loop(0, 10)
```

(좀 더 안전한 재귀함수 작성하기)

```
def loop(acc, i, n):
    if(i >= n):
        return acc
    else:
        loop(acc + i, i + 1, n)
loop(0, 0, 10)
```

(구문을 하나의 함수로, 간결하게 대체할 수 있다!)
(함수형 프로그래밍 패러다임!)

프로그래밍 기법 - 리스트

```
>>> lst1 = []
>>> lst2 = [1, 2, 3]
>>> lst2.append(4)
>>> lst2
[1, 2, 3, 4]
>>> lst2.append("hello!")
>>> lst1.append(["This is list in list"])
>>> lst1
[['This is list in list']]
```

멘델의 유전법칙

멘델의 유전법칙

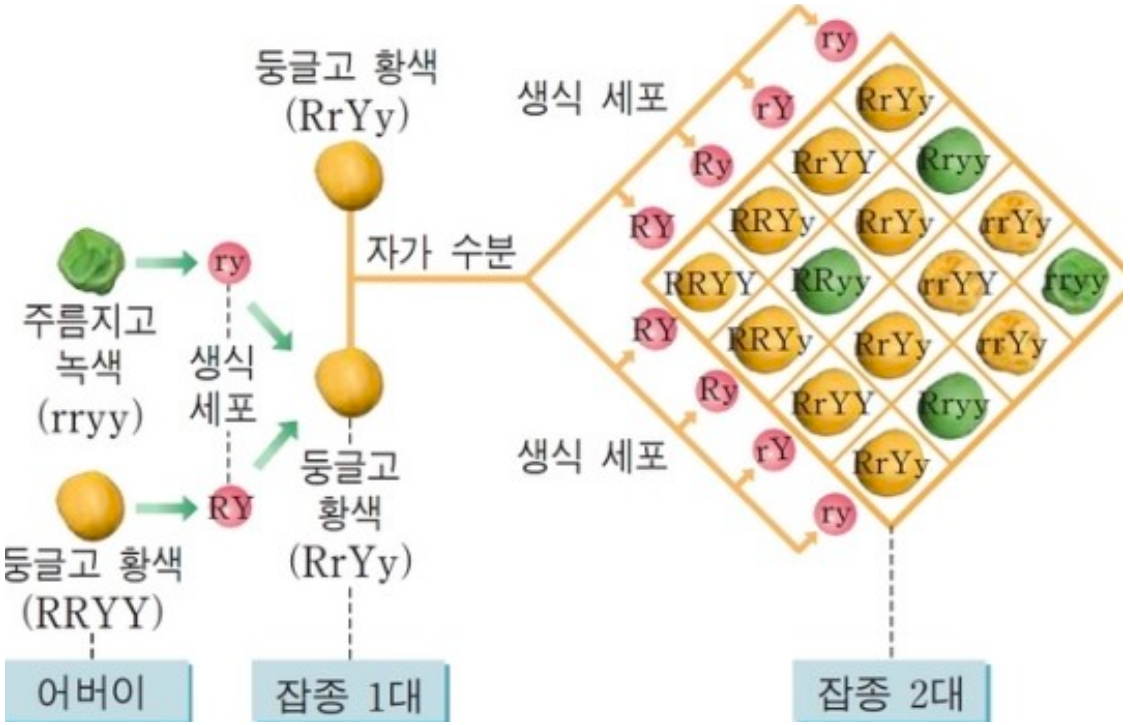
- 멘델의 유전 법칙은 멘델이 완두콩을 이용한 실험을 정리해 얻은 결과입니다.
- 우열의 원리 - 형질을 결정하는 유전자에는 우성/열성이 있어 이형접합인 경우 우성 유전자의 형질이 나타남
- 분리의 법칙 - 각각의 대립유전자는 나누어져 자손으로 전달된다.
- 독립의 법칙 - 각 형질에 관여하는 유전자는 독립적으로 움직인다.

멘델 유전법칙의 현대적 해석

- 우열의 원리 - 어떤 대사에 필요한 물질을 우성유전자가 만든다.
- 분리의 법칙 - 생식세포가 만들어질 때 염색분체가 분리된다
- 독립의 법칙 - 유전자는 다름 염색체 상에 존재하기 때문!

  (유전자가 연관이 아닌 경우)

퍼넷사각형

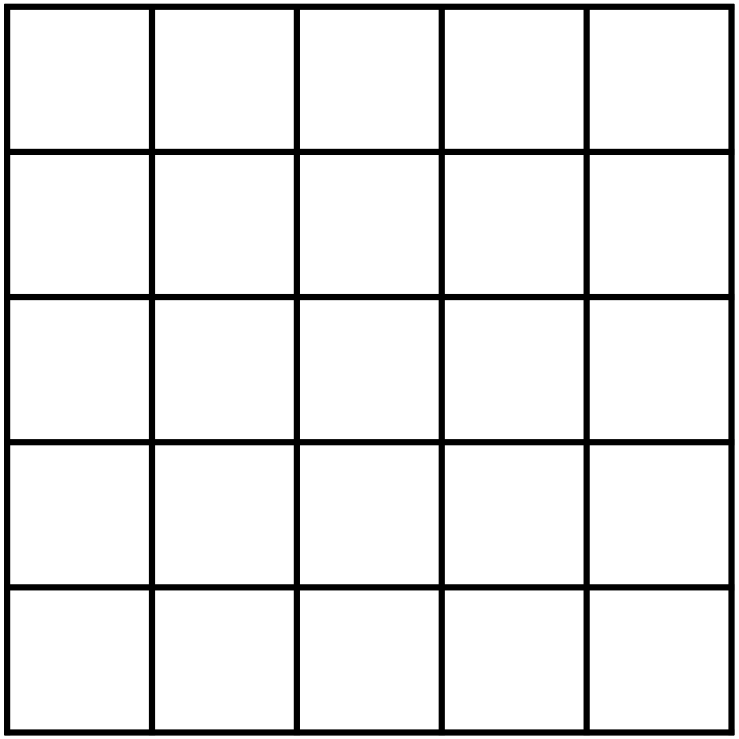


- 어떤 세대의 유전자형을 알 때, 다음 세대의 유전자형을 구하는 방법

직접 퍼넷사각형을 그려봐요!

- 손으로 직접 RrYy X RrYy 를 교배했을 때 나오는 자손을 모두 찾아 봅시다.

- 그럼 RrYyAaBb X RrYyAaBb 의 경우는 어떻게 될까요??



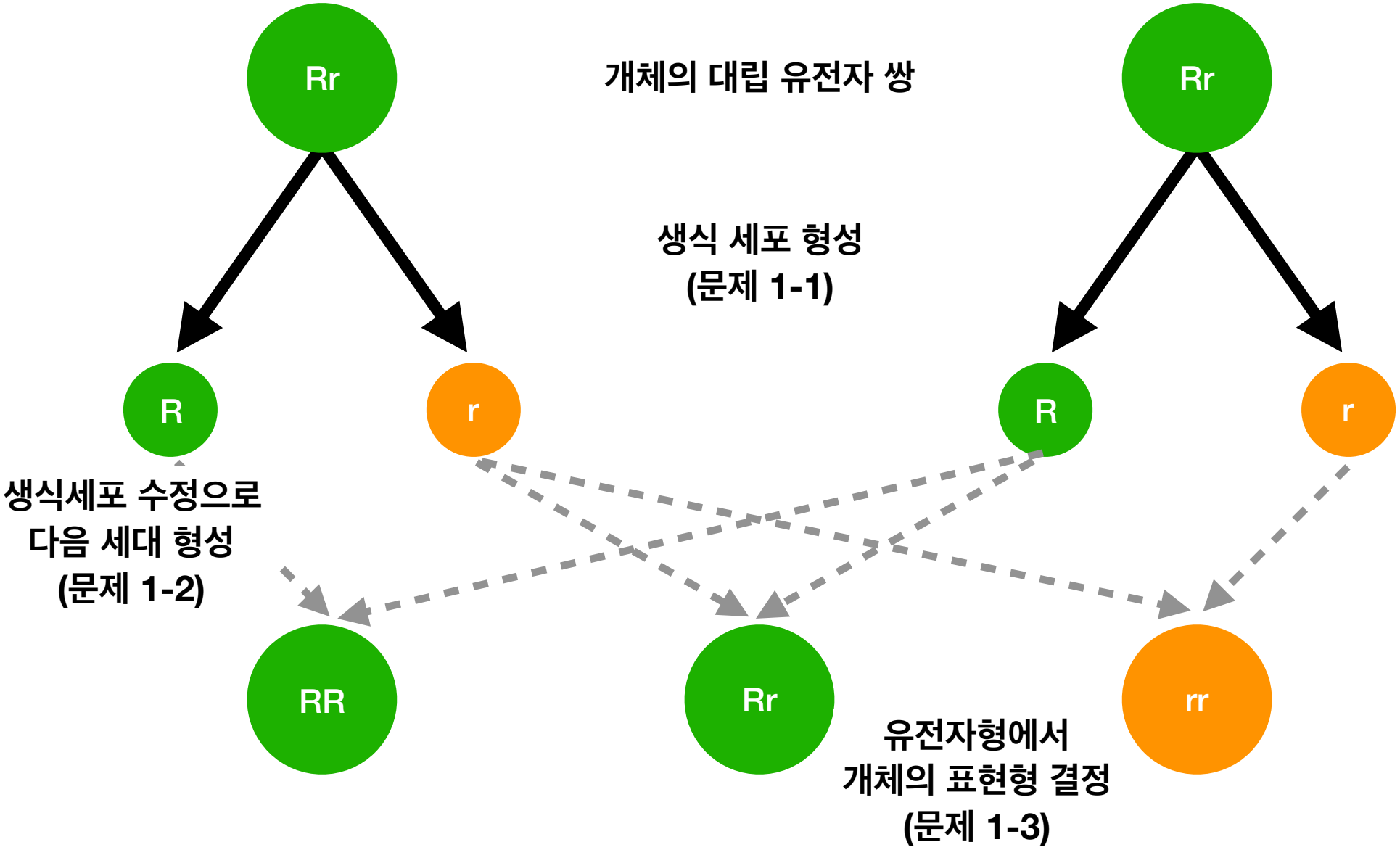
멘델의 유전법칙 적용하기

(대립유전자가 하나일 때)

유전자 흐름 모델링

- 어떤 개체가 “Rr”이라는 대립 유전자 쌍을 가지고 있다고 합시다.
- 생식세포가 만들어지면 대립유전자는 각각 “R”과 “r”로 나뉘어집니다.
- 각각의 생식세포 “R”과 “r”은 다른 “R”과 “r”을 만나 다시 “RR”, “Rr” “rr” 대립 유전자 쌍을 가진 개체가 됩니다.
- 어떻게 모델링 할 수 있을까요?

유전자 흐름 모델링


Rr
개체의 대립 유전자 쌍
Rr
생식 세포 형성
(문제 1-1)
R
r
R
r
생식세포 수정으로
다음 세대 형성
(문제 1-2)
RR
Rr
rr
유전자형에서
개체의 표현형 결정
(문제 1-3)

문제 1-1 유전자의 생식세포 만들기

- 유전자를 저장하는 구조를 string : “Rr” 로 취급합시다.
- string으로 나타난 이 유전자가 생식세포가 되었을 때 “R” 과 “r”이라는 유전자로 분리되도록 합니다.
- 이것을 처리하는 함수를 만들어 봅시다.

문제 1-1 유전자의 생식세포 만들기

```python
# Python3

# Gene is given by string
gene1 = "RR"
gene2 = "Rr"

# Make function that make "germ_cell"
# function : list(gene) -> list(germ_cell_gene)
def make_germ_cell(gene):
    # Fill with some code
    # You should return "list of germ_cell gene" like "R" and "r"
    pass
```

주어진 유전자형에서 발생할 수 있는 생식세포들을 담은 리스트를 리턴하는 make_germ_cell(gene) 을 완성해봅시다.

문제 1-1 유전자의 생식세포 만들기

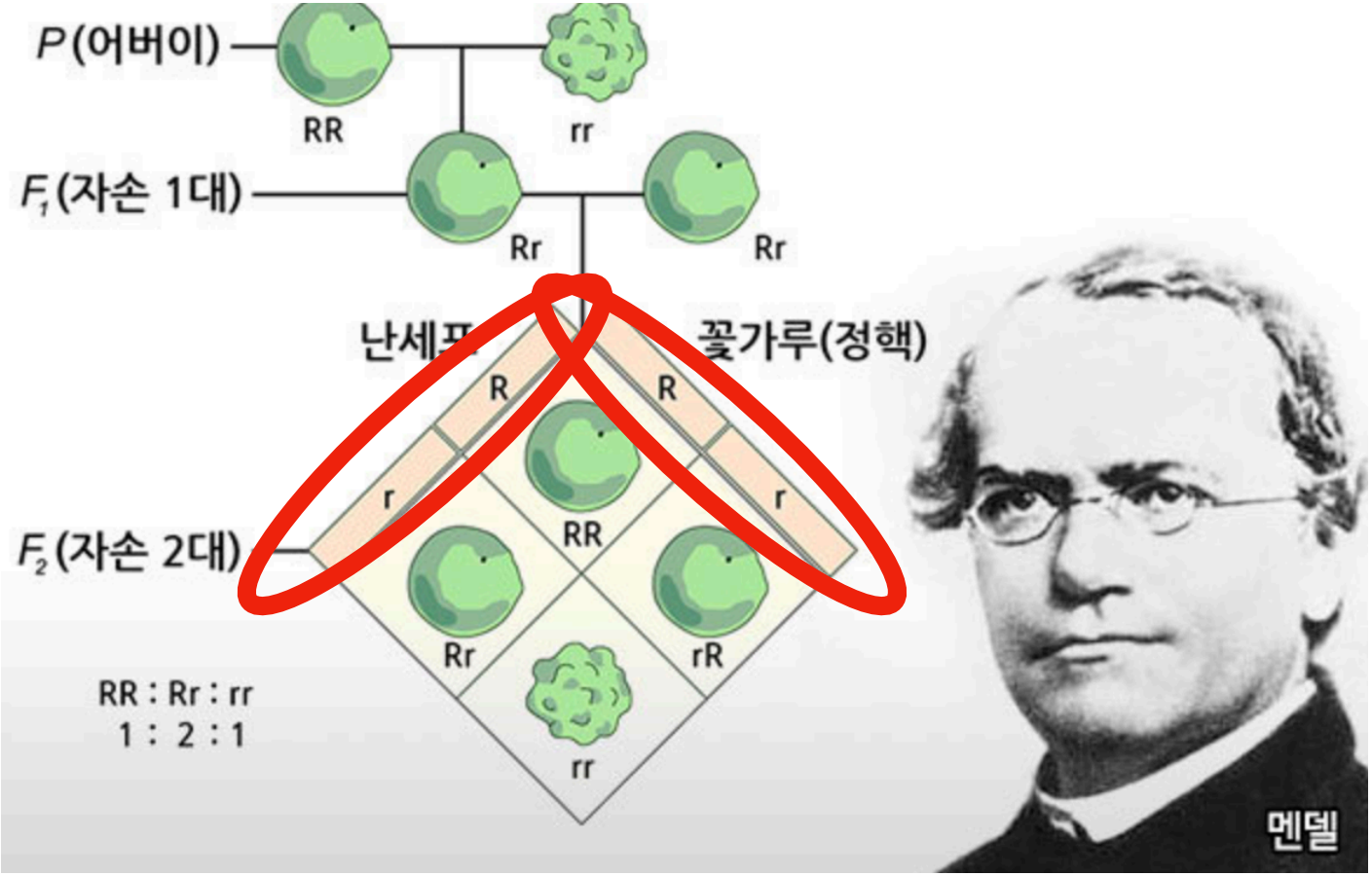
구현 결과

```
hwang@Seungmins-MBP mendel % python3 mendel_1.py
gene1 : RR
gene2 : Rr
----
RR -> ['R', 'R']
Rr -> ['R', 'r']
----
genotypes of next generation are :
['RR', 'Rr']
['RR', 'Rr']
```

문제 1-2 생식세포 수정 구현하기

- 유성생식에도 여러 종류가 있겠지만, 생식세포가 1 : 1로 만나 새로운 개체를 형성하는 방식으로 이루어진다고 제한합시다.
- 생식을 하는 두 개체로부터 “생식세포 유전자”리스트를 얻고
- 두 리스트 안의 생식 세포 조합으로 생겨날 수 있는 개체를 출력합니다.
- **“퍼넷 사각형”**을 이용합시다!

퍼넷사각형 다시 보기


P(어버이)
RR
rr
F1(자손 1대)
Rr
Rr
난세포
꽃가루(정핵)
R
R
r
r
F2(자손 2대)
RR
Rr
rR
rr
RR : Rr : rr
1 : 2 : 1
멘델

문제 1-2 생식세포 수정 구현하기

	R	r
R	RR	Rr
r	Rr	rr

이런 모양의 표를 만드는 것으로
생식세포가 수정해
새 개체를 만드는 것을
구현할 수 있습니다

new_gene[i][j] := germ1[i] and germ2[j]

다음 식을 통해 새 개체에서 나타날 유전자형을 결정하는 표를 작성할 수 있습니다.

문제 1-2 생식세포 수정 구현하기

```
def lst_mul(lst1, lst2):
    # matrix multiplication
    len1, len2 = len(lst1), len(lst2)
    matrix = []

    for i in range(len1):
        row_vec = []
        for j in range(len2):
            # ???
        # ???
    return matrix
```

germ_cell list를 이용해 앞의 슬라이드에 나왔던 빈칸을 채울 수 있는 코드를 작성해 봅시다.
(힌트 : “abc” + “def” -> “abcdef”를 사용하면 편하게 해결 가능!)

문제 1-2 생식세포 수정 구현하기

- 구현 결과

```
hwang@Seungmins-MBP mendel % python3 mendel_1.py
gene1 : RR
gene2 : Rr
----
RR -> ['R', 'R']
Rr -> ['R', 'r']
----
genotypes of next generation are :
['RR', 'Rr']
['RR', 'Rr']
```

문제 1-3 유전자형에서 표현형 결정하기

- 주어진 유전자형이 어떤 표현형을 나타낼 지 결정하는 함수를 작성해 봅시다.
- 예를 들어, RR, Rr의 경우 R으로, rr 의 경우 r로 대응시킬 수 있어야 합니다.
- 유전자형을 받아 표현형을 반환하는 phenotype 함수를 작성!
- 유전자형, 표현형 모두 2차원 배열 형태의 자료 구조를 지님!

문제 1-3 유전자형에서 표현형 결정하기

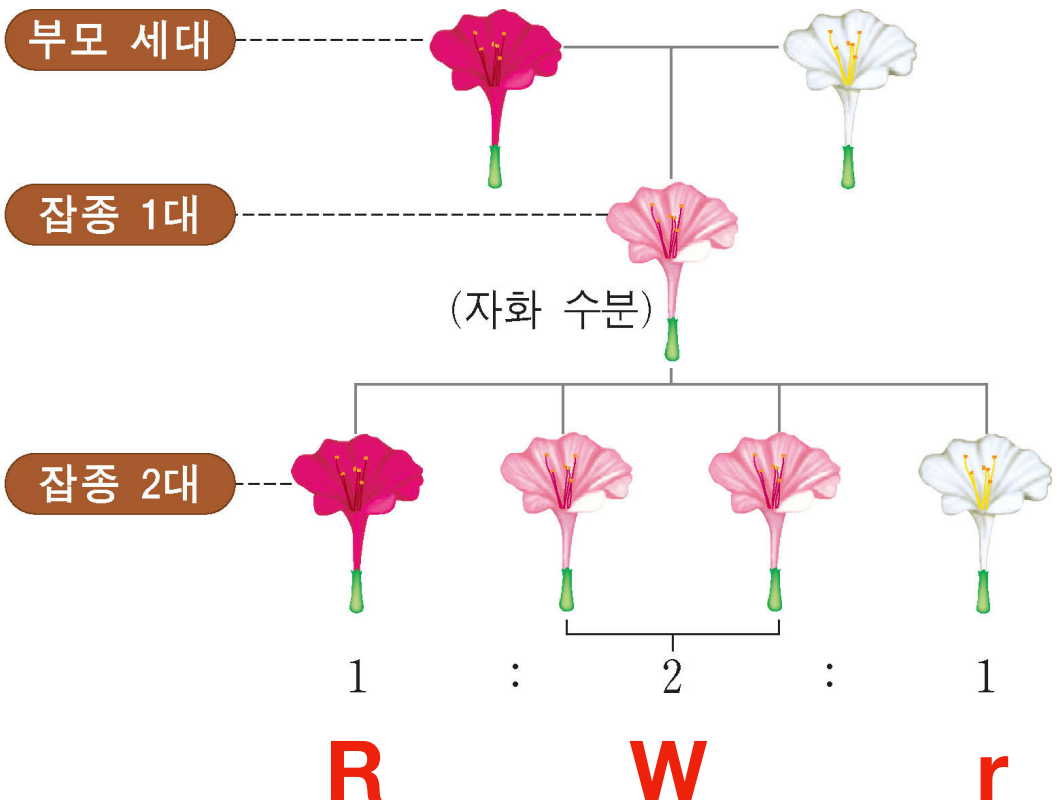
```
def get_phenotype(punnett):
    # find phenotypes from punnett square(genotype)
    pheonotype_table = []
    for row in punnett:
        new_row = []
        for gene in row:
            # ???
        pheonotype_table.append(new_row)
    return pheonotype_table
```

 # ??? 부분에 어떤 코드를 작성해야 할까요??
위쪽 코드에 사용된 반복문은, 아까 구한 유전자형의 2차원 리스트 원소
각각에 접근할 수 있습니다.

Rr 인 경우? RR인 경우? rr인 경우? 어떻게 처리를 하면 될까요?

문제 1-3 불완전 우성이 나타나는 경우?

- 중간형질도 대응시킬 수 있도록 프로그램을 수정해봅시다!



멘델의 유전법칙 적용하기

(대립유전자가 두 개 이상일 때)

문제 2 여러 대립 유전자가 있을 때

- 지금껏 해온 단일 대립 유전자 상황에서는 “간단하게” 특수한 상황임을 이용해 문제를 해결할 수 있습니다.
- 그렇다면 여러개의 대립 유전자가 동시에 존재할 때에도 작동하게 만들기 위해서는 어떤 방법으로 문제를 해결해야 할까요??
- 일반화를 위해 두 개의 대립유전자가 있는 상황을 생각합시다.

앞에서 해결한 문제 되돌아보기

- 문제 1-1 유전자 string으로부터 생식세포 생성
  - 알고리즘 수정 필요함.
- 문제 1-2 생식세포를 조합해 새로운 개체의 유전자 결정
  - 그대로 이용하되, 조금 수정(RyRY -> RRYy로 고치는 모듈)
- 문제 1-3 유전자형으로부터 개체의 표현형 알아내기
  - 표현형이 다양하므로, 수정할 필요가 있음

문제 2-1 여러 대립유전자의 생식세포

- 어떤 개체의 두 대립유전자 쌍이 RrYy라 해 봅시다.
- 두 대립유전자가 독립이라면
- RY, Ry, rY, ry
- 총 네 종류의 생식세포가 발생할 수 있음.

문제 2-1 알고리즘 연구

- 대립유전자 n개는 항상 한 쌍씩 묶여 n개가 나열된 형태라 가정
  - 예) RrYyAa : (Rr) (Yy) (Aa), 2개씩 떼어내 다룰 수 있음
- 각 대립유전자는 독립적으로 움직인다고 가정

문제 2-1 알고리즘 연구
동전 문제 생각해보기

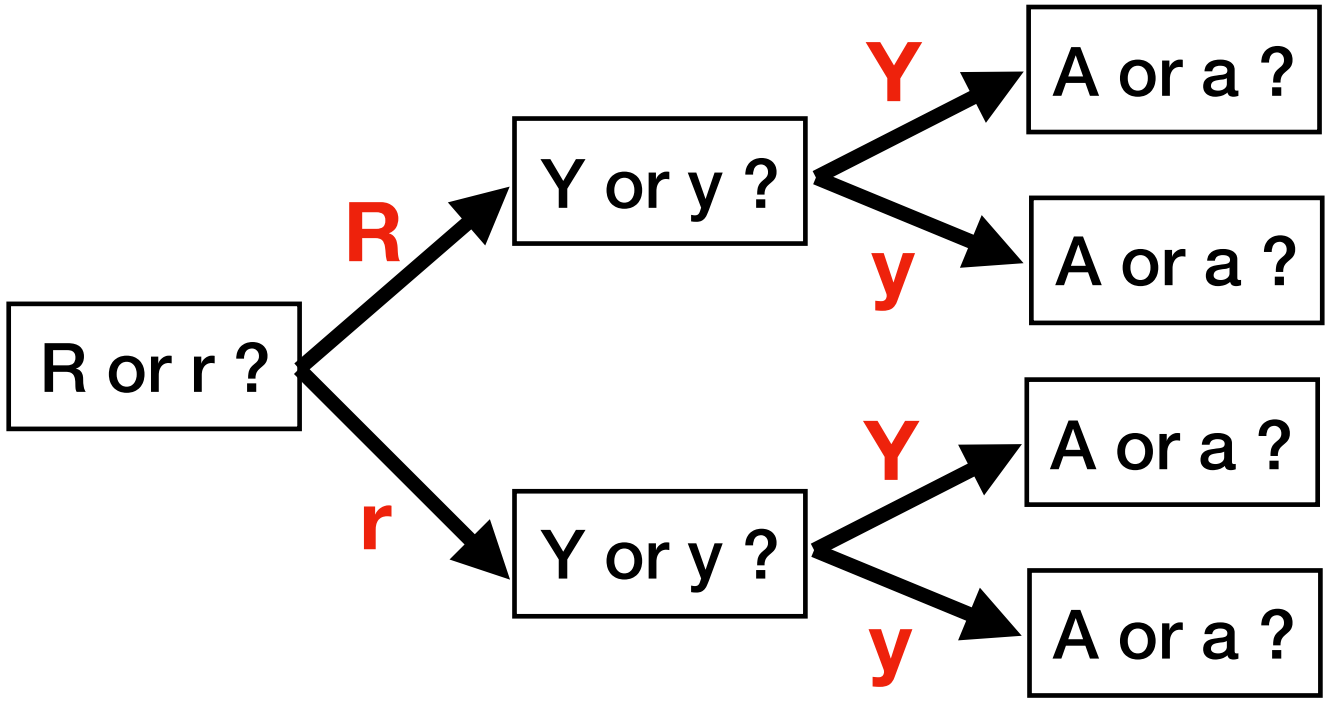
- 간단한 확률 문제에서 생각해보기
  - 500원 동전, 100원 동전, 50원 동전이 있습니다.
  - 동전들을 던져서 나온 동전들의 면을 확인할 때(순서 고려 X)
  - 총 몇 가지의 경우가 존재할까요??

문제 2-1 알고리즘 연구
동전 문제 생각해보기

- 간단한 확률 문제에서 생각해보기(답안)
  - 500 앞 100 앞 50 앞 (1가지)
  - 500 앞 100 앞 50 뒤 (2가지)
  - 500 앞 100 뒤 50 앞 (3가지)
  - …
  - 500 뒤 100 뒤 50 뒤 (8가지)

문제 2-1 알고리즘 연구

Rr Yy Aa

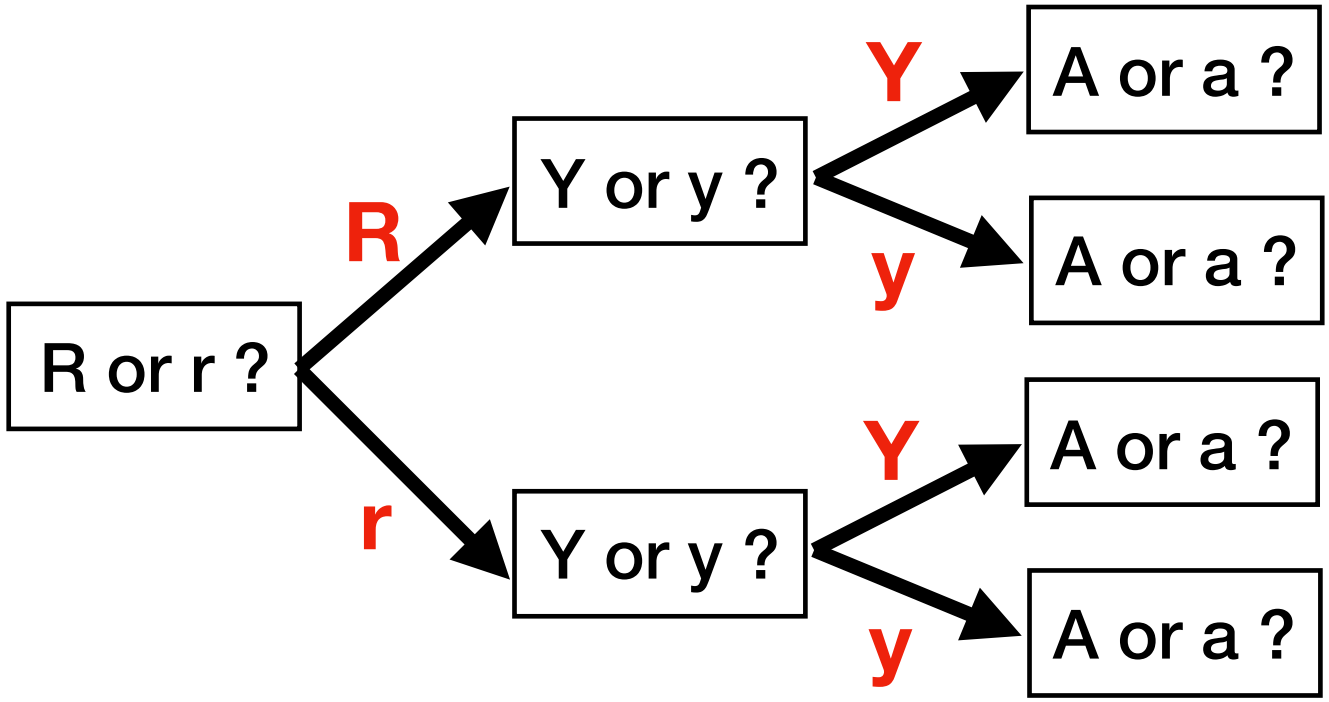


(많아서 조금 생략합니다)
수형도를 그려봅니다!

- 재귀적인 형태가 보인다! 잘 안보인다고요??

문제 2-1 알고리즘 연구

(Rr) (Yy) (Aa)



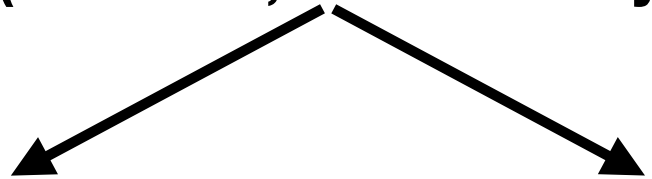
(많아서 조금 생략합니다)
수형도를 그려봅니다!

- 각 대립유전자 블록을 매 단계 하나씩 넣어주고
- (a, b) 중 a를 넣은 것과, b를 넣은 것을 각각 재귀호출…?

문제 2-1 알고리즘 연구

loop(acc = “R”, remain = “YyAa”)

loop(acc = “r”, remain = “YyAa”)



loop(acc = “rY”, remain = “Aa”)

loop(acc = “rY”, remain = “Aa”)

loop(acc = “ry”, remain = “Aa”)

loop(acc = “ry”, remain = “Aa”)

- 대략 이런 느낌으로 설계할 수 있습니다.
- 매 유전자 블록마다 둘 중 하나를 선택해, 선택 결과를 누적시키고
- 남은 유전자가 없을 때 누적된 결과를 리스트에 추가, 그리고 종료

문제 2-1 구현해보기

- 알고리즘을 거의 다 알려드렸으니, 스스로 한 번 구현을 시도해 보세요!
- 추가로 힌트를 드리자면
- Tail-recursion에 대해 찾아보세요!
- 그래도 어렵다면 저에게 코드 스켈레톤 제공을 요청하세요!

문제 2-1 구현해보기

```
def make_germ_cell(gene):
    # init germ cell
    # gene is given by : (Rr)(Yy)(Aa)...
    germ_cell_list = []

    # Higher order function
    def loop(acc, remain):
        if (remain == ""):
            germ_cell_list.append(acc)
        else:
            hd_gene = remain[:2]
            remain = remain[2:]
            loop(acc + hd_gene[0], remain)
            loop(acc + hd_gene[1], remain)
    loop("", gene)

    print("genotypes from ", gene, " are : \n", germ_cell_list)
    return germ_cell_list
```

문제 2-1 구현 결과

```
hwang@Seungmins-MBP mendel % python3 mendel_2.py
gene :  RrYy  generated
gene :  RrYy  generated
----
genotypes from  RrYy  are :
 ['RY', 'Ry', 'rY', 'ry']
genotypes from  RrYy  are :
 ['RY', 'Ry', 'rY', 'ry']
----
gene :  RrYyAa  generated
gene :  RrYyAa  generated
----
genotypes from  RrYyAa  are :
 ['RYA', 'RYa', 'RyA', 'Rya', 'rYA', 'rYa', 'ryA', 'rya']
genotypes from  RrYyAa  are :
 ['RYA', 'RYa', 'RyA', 'Rya', 'rYA', 'rYa', 'ryA', 'rya']
----
```

문제 2-2 생식세포 수정 모델링

- 생식세포 수정 모델링은, 행렬 곱으로 구현했는데
- 거의 수정할 필요가 없습니다.
- 다만, 기존 행렬 곱을 사용했을 때 유전자가
  RyArYa 와 같은 순서로 배열된 것을
  RrYyAa로 바꿔줘야합니다!
- 이 부분만 추가시켜 기능을 완성하도록 합시다!

문제 2-2 알고리즘 연구

- 대립유전자가 n쌍 있는 경우를 보면
- 길이 2n인 유전자에서 n + n 구조를 이루고 있음
- 그렇다면 각각의 유전자를 동시에 한 글자씩 읽어가면서
- 각각에서 읽어들인 두 글자의 대소문자를 비교해
- 하나의 새로운 유전자로 합치면 될 것!

문제 2-2 구현해보기

문제 2-2 구현 결과

```
gene :  RrYyAa  generated
gene :  RrYyAa  generated
----
genotypes from  RrYyAa  are :
 ['RYA', 'RYa', 'RyA', 'Rya', 'rYA', 'rYa', 'ryA', 'rya']
genotypes from  RrYyAa  are :
 ['RYA', 'RYa', 'RyA', 'Rya', 'rYA', 'rYa', 'ryA', 'rya']
----
['RRYYAA', 'RRYYAa', 'RRYyAA', 'RRYyAa', 'RrYYAA', 'RrYYAa', 'RrYyAA', 'RrYyAa']
['RRYYAa', 'RRYYaa', 'RRYyAa', 'RRYyaa', 'RrYYAa', 'RrYYaa', 'RrYyAa', 'RrYyaa']
['RRYyAA', 'RRYyAa', 'RRyyAA', 'RRyyAa', 'RrYyAA', 'RrYyAa', 'RryyAA', 'RryyAa']
['RRYyAa', 'RRYyaa', 'RRyyAa', 'RRyyaa', 'RrYyAa', 'RrYyaa', 'RryyAa', 'Rryyaa']
['RrYYAA', 'RrYYAa', 'RrYyAA', 'RrYyAa', 'rrYYAA', 'rrYYAa', 'rrYyAA', 'rrYyAa']
['RrYYAa', 'RrYYaa', 'RrYyAa', 'RrYyaa', 'rrYYAa', 'rrYYaa', 'rrYyAa', 'rrYyaa']
['RrYyAA', 'RrYyAa', 'RryyAA', 'RryyAa', 'rrYyAA', 'rrYyAa', 'rryyAA', 'rryyAa']
['RrYyAa', 'RrYyaa', 'RryyAa', 'Rryyaa', 'rrYyAa', 'rrYyaa', 'rryyAa', 'rryyaa']
```

- 다음과 같이 모든 조합이 다 출력되는 모습을 확인할 수 있습니다.

문제 2-3 유전자형에서 표현형 결정하기

- 앞에서 해결했던 문제와 상당히 유사합니다.
- 다만, 대립유전자가 여러 개 섞여 있기 때문에, 표현형도 여러개
- 유전자형을 두 글자씩 읽어, 각각 표현형을 결정할 수 있도록 만들어 봅시다!
- 예시 > (Rr) (Yy) (aa) -> (R) (Y) (a)

문제 2-3 구현 결과

```
____
['RYA', 'RYa', 'RyA', 'Rya', 'rYA', 'rYa', 'ryA', 'rya']
['RYa', 'RYa', 'Rya', 'Rya', 'rYa', 'rYa', 'rya', 'rya']
['RyA', 'Rya', 'RyA', 'Rya', 'ryA', 'rya', 'ryA', 'rya']
['Rya', 'Rya', 'Rya', 'Rya', 'rya', 'rya', 'rya', 'rya']
['rYA', 'rYa', 'ryA', 'rya', 'rYA', 'rYa', 'ryA', 'rya']
['rYa', 'rYa', 'rya', 'rya', 'rYa', 'rYa', 'rya', 'rya']
['ryA', 'rya', 'ryA', 'rya', 'ryA', 'rya', 'ryA', 'rya']
['rya', 'rya', 'rya', 'rya', 'rya', 'rya', 'rya', 'rya']
```

- 다음과 같이 punnett 사각형에서 나타나는 개체의 표현형을 결정!

추가로 해 볼만한 과제들?

추가로 해 볼 만한 과제들?

- 상 염색체 유전이 아닌 성 염색체라면…?
- 감수 1분열에서 교차가 일어난다면…?
- 표현형이 결정되는 방식이 다른 유전이라면…?
- 세대를 거듭하면서 어떤 집단의 유전자 비율이 어떻게 되는지…?

심화학습

뒤에서 다룰 내용은...

- 만약 시간이 부족하다면 다루지 않을 예정입니다!
- 그러나, 앞의 내용을 충분히 이해하고
- Class 사용에 능숙해진다면 꼭 한 번 도전해보세요!!

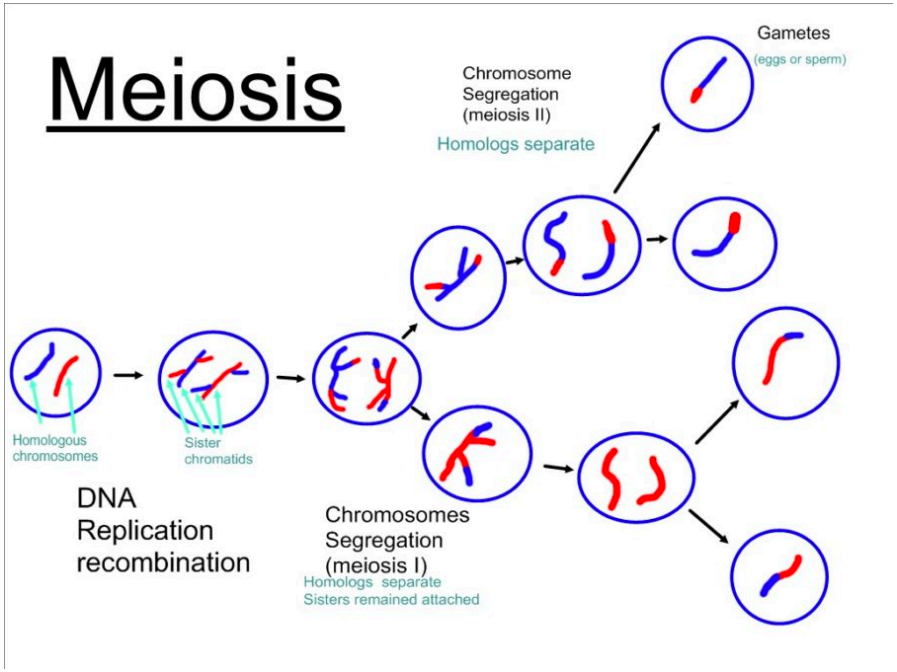
염색체 기반 유전현상 모델

(OOP를 이용한 염색체, 유전자 모델링)

새로운 모델의 필요성

- 멘델의 유전법칙을 기반으로 다음 세대의 자손이 어떻게 될 지 살펴볼 수 있었습니다.
- 그러나, 이것은 몇 가지 종류의 유전 현상을 “관찰”한 것 뿐이고
- 실제로 생체 내에서 일어나는 일에 대해 다룬 것은 아닙니다!
- 뿐만 아니라, 이후 알려진 여러 예외상황을 다루기 어렵게 합니다

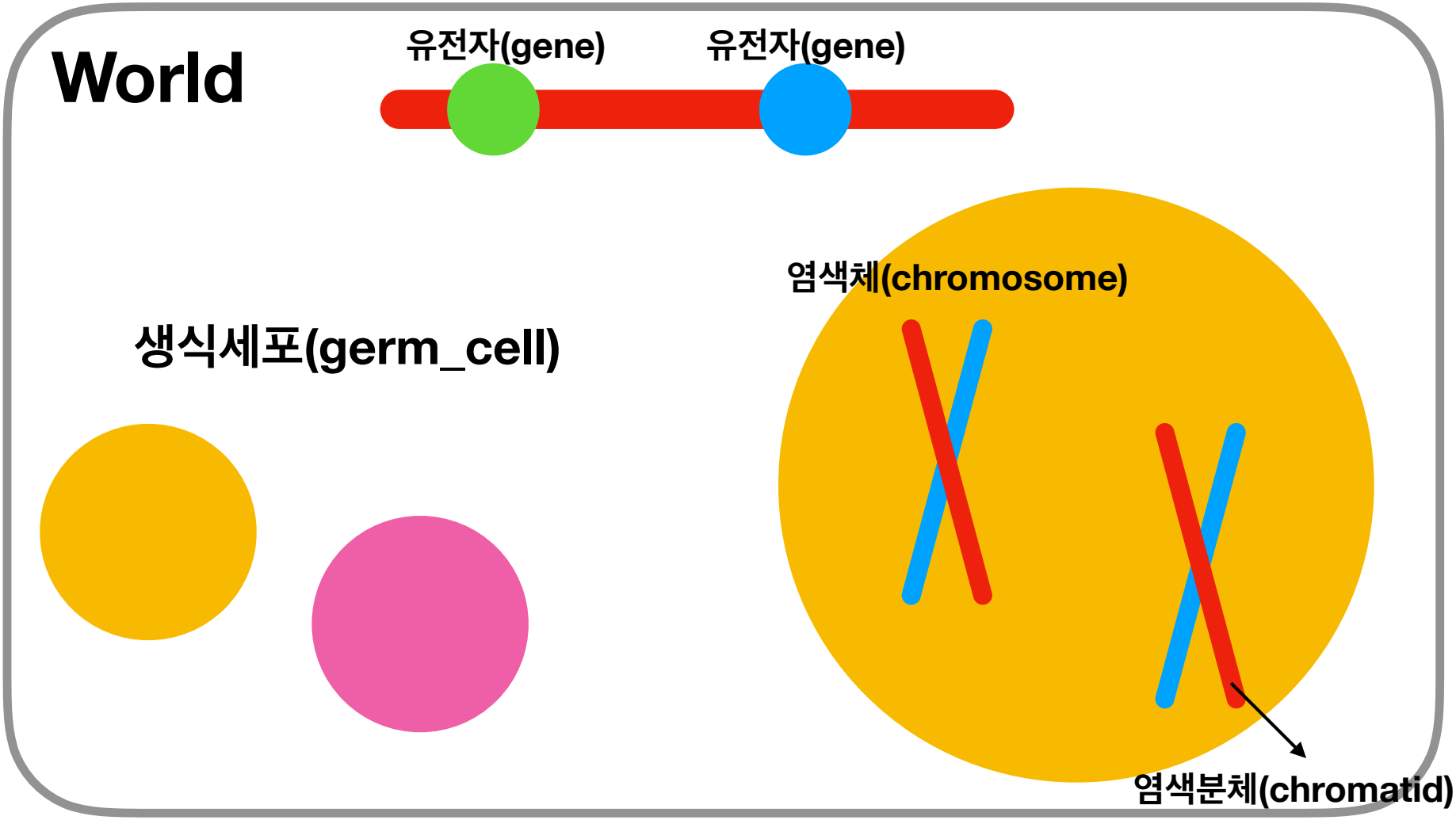
미시적 관점에서 본 유전 현상



- 감수 분열을 통한 생식세포 형성 (1배체)
- 생식 세포간의 수정
- 수정을 통해 서로 다른 염색분체를 받아들임, 다시 2배체가 됨!

문제 3 염색체 기반 유전현상 모델링

- 이를 모델링하기 위해 우리는 어떤 객체(Object)를 도입해야 할까?



문제 3 염색체 기반 유전현상 모델링

1. World - 여러 종류의 생식세포들이 있고, 수정이 이루어 질 수 있는 환경
2. 생식세포 - 여러 염색체를 담고 있는 컨테이너
3. 염색체 - 하나 또는 두 개의 염색분체를 가지고 있는 컨테이너
4. 염색분체 - 여러 유전자를 가지고 있는 컨테이너
5. 유전자 - 유전자의 코드(ex. R, Y …)를 보유하고 있는 객체

5. 유전자 Object 구현

```
# gene.py
class gene():
    # 유전자를 나타내는 class
    def __init__(self, code = "None"):
        self.code = code
        print("gene [", code, "] has been generated")
    def __str__(self):
        return self.code
```

- 유전자가 가져야 할 속성
  - 유전자 코드 (R? r? Y? A?? 등의 알파벳 기호)
  - 세부 속성을 설명하는 줄글…?(optional)

4. 염색분체 Object 구현

```
class chromatid():
    def __init__ (self, gene_list = []):
        self.genes = gene_list

    def add(self, gene):
        self.genes.append(gene)

    def remove(self, gene):
        self.genes.remove()
```

- 염색 분체가 가져야 할 속성?
    - 염색분체가 가지고 있는 유전자들이 무엇인지??