

# 기초 파이썬

최적화를 위한 파이썬



# 파이썬



1. 개발 환경 설정
2. 자료형
3. 연산자
4. 조건문/반복문
5. `pandas`
6. PuLP 라이브러리
7. PuLP - LP 모델 구축

# 파이썬 가상환경 ANACONDA



다운로드  
설치 안내

# 자료형

	자료형	설명	변경 가능성	저장 모델
수치형	int / float	-1,0,1,2,3.14,0.15	불가능	Literal
문자열	str	'Hello World', "123"	불가능	Literal
튜플	tuple	() (1,2,3)	불가능	Container
리스트	list	[] [1,2,3]	가능	Container
사전	dict	{key:value} {'name':'FIELD'}	가능	Container

```
>>> type(3)
<class 'int'>
>>> type(3.1415)
<class 'float'>
>>> type("FIELD is fantastic")
<class 'str'>
>>> type(3 + 7j)
<class 'complex'>
>>> type(True)
<class 'bool'>
```

정수: int

실수: float

문자열: str

복소수: complex

논리값: bool

1

정수 (Integer)  
int() : **2023**

2

실수 (Float)  
float() : **2.023**

3

문자열 (String)  
str() : **"FIELD"**

4

복소수 (Complex)  
complex() : **3 + 7j**

5

논리값 (Boolean)  
bool() : **True / False**



```
x = [10, 'a', 20]
```

```
x
```

```
[10, 'a', 20]
```

## list[] : 가변형

- [] 사용
- 저장 자료형 제한 없다
- 요소 구분은 ,로



```
y = (10, 'a', 30)
```

```
y
```

```
(10, 'a', 30)
```

## tuple() : 불가변형

- [] (대괄호)가 아닌 () (괄호) 사용
- 내부 값 추가, 변경, 삭제 불가



```
dic = {'name': 'field', 'year': '2023'}
```

```
dic
```

```
{'name': 'field', 'year': '2023'}
```

## dict{} : 가변형

- {} 중괄호 사용
- {key:value}로 묶임

# 연산자

```
❌ □ —
** 는 거듭제곱 연산입니다. (a ** b = ab )
>>> 2**16
65536

% 는 나눗셈의 나머지 연산입니다.
>>> 7 % 3
1

// 는 정수 나눗셈(소수 부분을 제외한 나눗셈) 연산입니다.
>>> 13.0 // 4.0
3.0
>>> 9 / 7
1.2857142857142858
< >
```

```
❌ □ —
>>> 3 < 5
True
>>> 27 == 14 ← '=='은 두 값이 같은지 비교하는 연산
False          대입 연산자인 = 와 헷갈리지 마세요
>>> 3.14 != 3.14
False
>>> 3.14 >= 3.14
True
>>> "Cheong" < "Choe"
True
>>> "3" == 3
False
< >
```

## 연산자

- ① 덧셈, 뺄셈 (+, -)
- ② 곱셈, 나눗셈 (\*, /, //, %)
- ③ 거듭제곱 (\*\*)

## 논리연산자

- ① ==, !=, >, <, <=, >=
- ② not, and, or

```
year = 2023
```

```
if year > 2023:  
    print("미래")  
elif year < 2023:  
    print("과거")  
else:  
    print("현재")
```

현재

## if 조건문: 수행문

- if 조건문 시 들여쓰기
- 조건문 다음에는 :(콜론) 사용
- 추가 조건문: `elif`
- 둘다 아닌 경우 : `else`



```
for i in range(5):  
    print(i)
```

0  
1  
2  
3  
4

## for 변수 in 리스트: 수행문

- 반복 가능한 객체 :  
리스트, 튜플, 문자열 등
- 반복가능객체 원소 개수만큼 반복

# PANDAS

자료구조 및 데이터 분석/처리를 위한 파이썬 패키지

## Series

1차원 데이터 구조  
index와 value의 형태

## Data Frame

행과 열로 이루어진  
표 형식의 데이터 구조

# Series

```
import pandas as pd
```

```
# Series 정의하기
```

```
obj = pd.Series([4, 7, -5, 3])
```

```
obj
```

index	value
0	4
1	7
2	-5
3	3

dtype: int64

index는 기본값으로  
0,1,2,.. 자동으로 생성

```
# python의 dictionary 자료형을 Series로 변환
```

```
obj.values
```

```
temp = {'d': 4, 'b': 7, 'a': -5, 'c': 3}
```

```
# dictionary의 key가 Series의 index
```

```
obj3 = pd.Series(temp)
```

```
obj3
```

index	value
d	4
b	7
a	-5
c	3

dtype: int64

In[] :

```
# Series의 값만 확인하기  
obj.values
```

Out[] :

```
4  
7  
-5  
3
```

```
# Series의 인덱스만 확인하기  
obj.index
```

```
0  
1  
2  
3
```

```
# Series의 자료형 확인하기  
obj.dtypes
```

int 64

```
# 인덱스를 바꾸기  
obj2 = pd.Series([4, 7, -5, 3], index=['d', 'b', 'a', 'c'])  
obj2
```

```
index value  
d      4  
b      7  
a     -5  
c      3  
dtype: int64
```

# Data Frame

```
import pandas as pd
```

```
# DataFrame 정의하기
```

```
df = pd.DataFrame({ "a" : [4, 5, 6], "b" : [7, 8, 9], "c" : [10, 11, 12]}, index = [1, 2, 3])
```



	a	b	c
1	4	7	10
2	5	8	11
3	6	9	12

# 데이터 조회

## DATAFRAME 열 선택 및 조작

- 데이터 프레임 명 [ "열 이름1", "열 이름2" ]
- 데이터 프레임명.열이름

# 특정 열에 특정 값 대입

데이터 프레임 명 ["열이름"] = value

데이터 프레임 명 ["열이름"] = [value1, value2,...]

## DataFrame 행 선택 및 조작

- 데이터 프레임 명 [ 시작 인덱스 : 끝 인덱스 ]

# 특정 행에 특정 값 대입

loc 이나 iloc의 메서드를 사용해서 특정 값을 대입

# 데이터 조회 (loc/iloc)

데이터 프레임 명.**loc**[ :, : ]  
**LABEL**을 통해 값을 찾는다.  
# : 모든 값을 의미

# 인덱스 명이 two인 행을 추출  
`df.loc["two"]`

# 콤마 이전은 행구간/콤마 이후는 열구간 설정  
#인덱스명이 two에서 four까지 point열과  
score열을 추출

`df.loc["two":"four", [ "point", "score" ]]`

데이터 프레임 명.**iloc**[ :, : ]  
**integer position**를 통해  
행과 열의 번호로 값을 찾는다

# index 번호를 사용하여 4번째의 행을 추출  
`df.iloc[3]`

# 행의 1,2,4번의 열 2,3번을 추출  
`df.iloc[ [0, 1, 3 ], [ 1, 2 ] ]`

# 데이터 조회 (boolean Indexing)

#"year" 열의 값이 2014보다 큰 data 조회  
`df[ df["year"] > 2014 ]`

#names가 kim인 행에 대하여 names열과 points열 조회  
`df[ df["names"] == "kim", ["names","points"] ]`

#두가지 조건을 만족하거나(&) 둘중 하나(|)를 만족하는 행을 조회하는 경우  
`df.loc[ 조건문1 & 조건문2 , : ]`  
`df.loc[ 조건문1 | 조건문 2 , : ]`

#조회한 행을 이용하여 새로운 값을 대입  
`df.loc[ 조건문, "열이름"] = value`



# DataFrame 함수들

```
df.sort_values("열이름")
```

열을 기준으로 정렬을 수행

```
df.rename(columns = {"기존열이름" : "새로운 열이름"})
```

기존의 열이름을 새로운 열이름으로 변경

```
df.drop(columns = ["열이름1", "열이름2"])
```

선택한 열 전체를 삭제

# Summarize 함수들

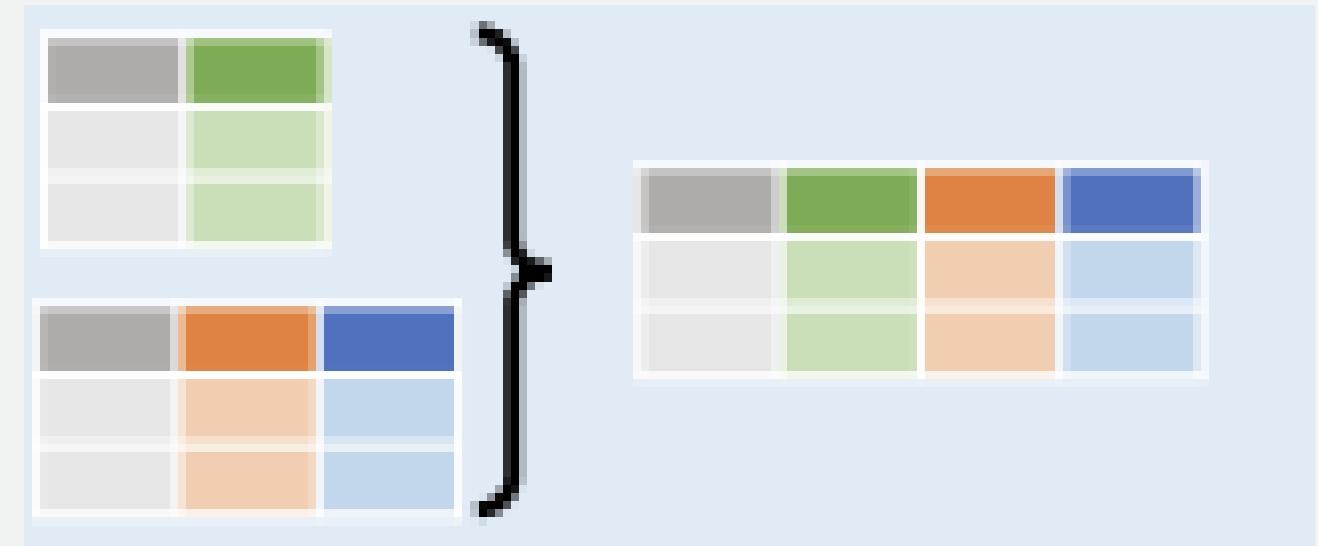
<code>sum()</code>	성분의 합을 계산	<code>var()</code>	성분의 분산을 계산
<code>count()</code>	성분의 (Nan이 아닌)값의 갯수를 계산	<code>std()</code>	성분의 표준편차를 계산
<code>median()</code>	성분의 중간값을 반환	<code>len(df)</code>	데이터 프레임의 행 갯수를 반환
<code>mean()</code>	성분의 평균을 계산	<code>df.shape</code>	데이터 프레임의 행과열을 반환
<code>min()</code>	성분의 최솟값을 계산	<code>df.head(n)</code>	데이터 프레임 처음 n개 행 반환
<code>max()</code>	성분의 최댓값을 계산	<code>df.tail(n)</code>	데이터 프레임 마지막 n개 행 반환
<code>argmin()</code>	성분의 최솟값이 위치한 인덱스 반환	<code>argmax()</code>	성분의 최댓값이 위치한 인덱스 반환

# Concat

#**행** 기준으로 데이터 프레임 연결  
`pd.concat([df1, df2])`



#**열** 기준으로 데이터 프레임 연결  
`pd.concat([df1, df2], axis=1)`



# merge

adf			bdf		
x1	x2		x1	x3	
A	1	+	A	T	=
B	2		B	F	
C	3		D	T	

x1	x2	x3
A	1	T
B	2	F
C	3	NaN

x1	x2	x3
A	1.0	T
B	2.0	F
D	NaN	T

x1	x2	x3
A	1	T
B	2	F

x1	x2	x3
A	1	T
B	2	F
C	3	NaN
D	NaN	T

# left 조인 on=합치는 기준

```
pd.merge(adf, bdf, how='left', on='x1')
```

adf의 행을 x1 열을 기준으로 일치하는 행을 bdf의 해당 행과 결합  
일치하는 행이 없는 경우에는 bdf의 열에는 NaN 값

# right 조인

```
pd.merge(adf, bdf, how='right', on='x1')
```

bdf의 행을 x1 열을 기준으로 일치하는 행을 adf의 해당 행과 결합  
일치하는 행이 없는 경우에는 adf의 열에는 NaN 값

# inner 조인

```
pd.merge(adf, bdf, how='inner', on='x1')
```

adf와 bdf의 x1 열이 일치하는 행만을 선택하여 두 데이터프레임을 조인  
일치하는 행이 없는 경우 해당 행은 결과에서 제외

# outer 조인

```
pd.merge(adf, bdf, how='outer', on='x1')
```

adf와 bdf의 x1 열을 기준으로 두 데이터프레임을 조인  
일치하는 행이 있는 경우에는 해당 행을 결합하고, 일치하지 않는 행은 NaN 값



## 데이터 불러오기

# pandas 사용하기

```
import pandas as pd
```

# 한글이 포함된 (csv)파일 불러오기

```
df = pd.read_csv('파일명.csv',  
encoding = "euc-kr")
```



## 데이터 내보내기

#csv 파일로 내보내기

```
변수.to_csv('파일주소/파일이름.csv')
```

#데이터프레임 index 제거하고 내보내기

```
df.to_csv('파일주소/파일이름.csv',  
index = False)
```



# PuLP

Python의 LP 모듈러



# download



```
pip install pulp
```



```
import pulp
```



# PuLP를 활용하여 LP를 푸는 법

1

변수(variable) 정의

2

문제(problem) 정의

3

목적 함수(objective function) 정의

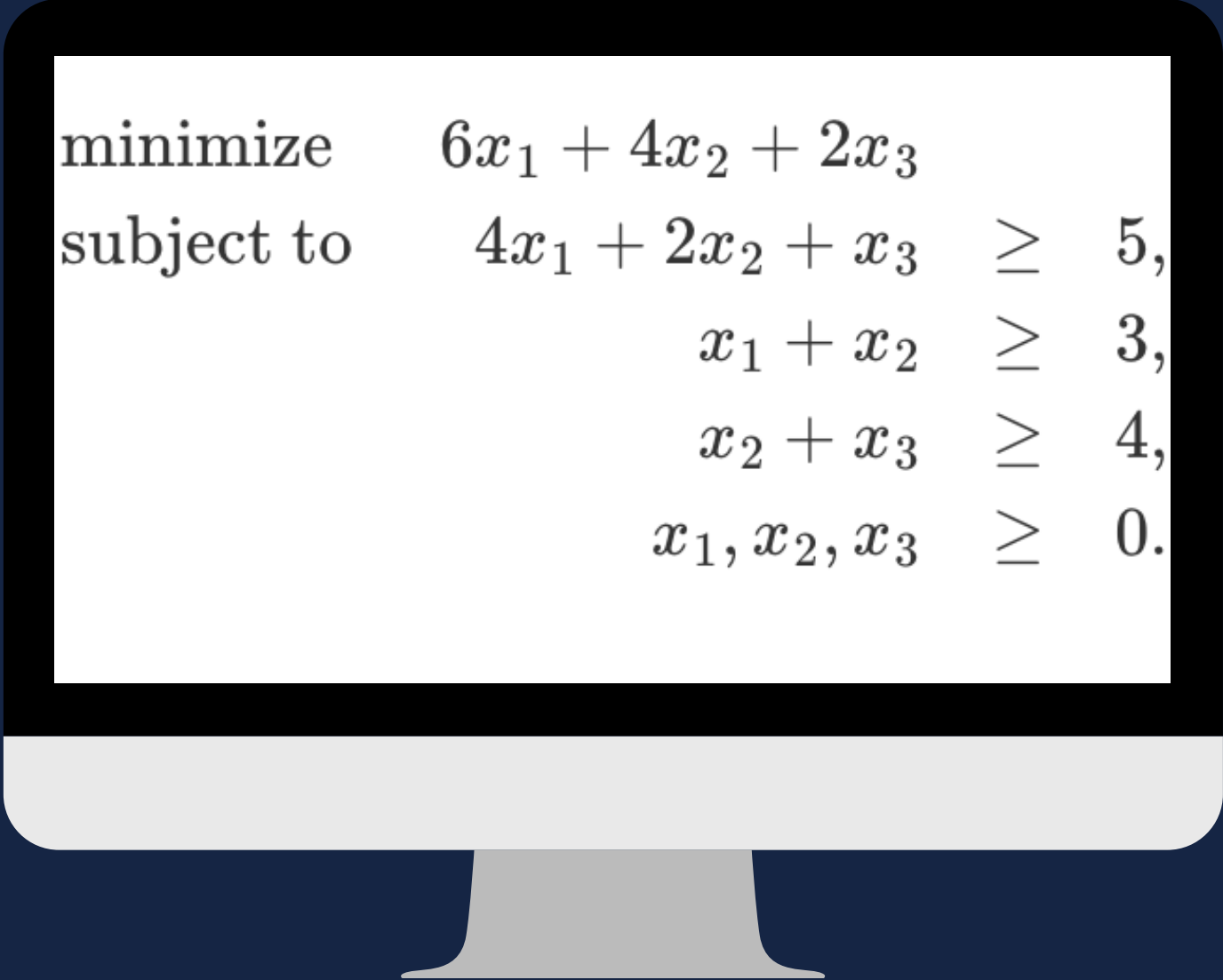
4

제약 조건(constraints) 정의

5

solve

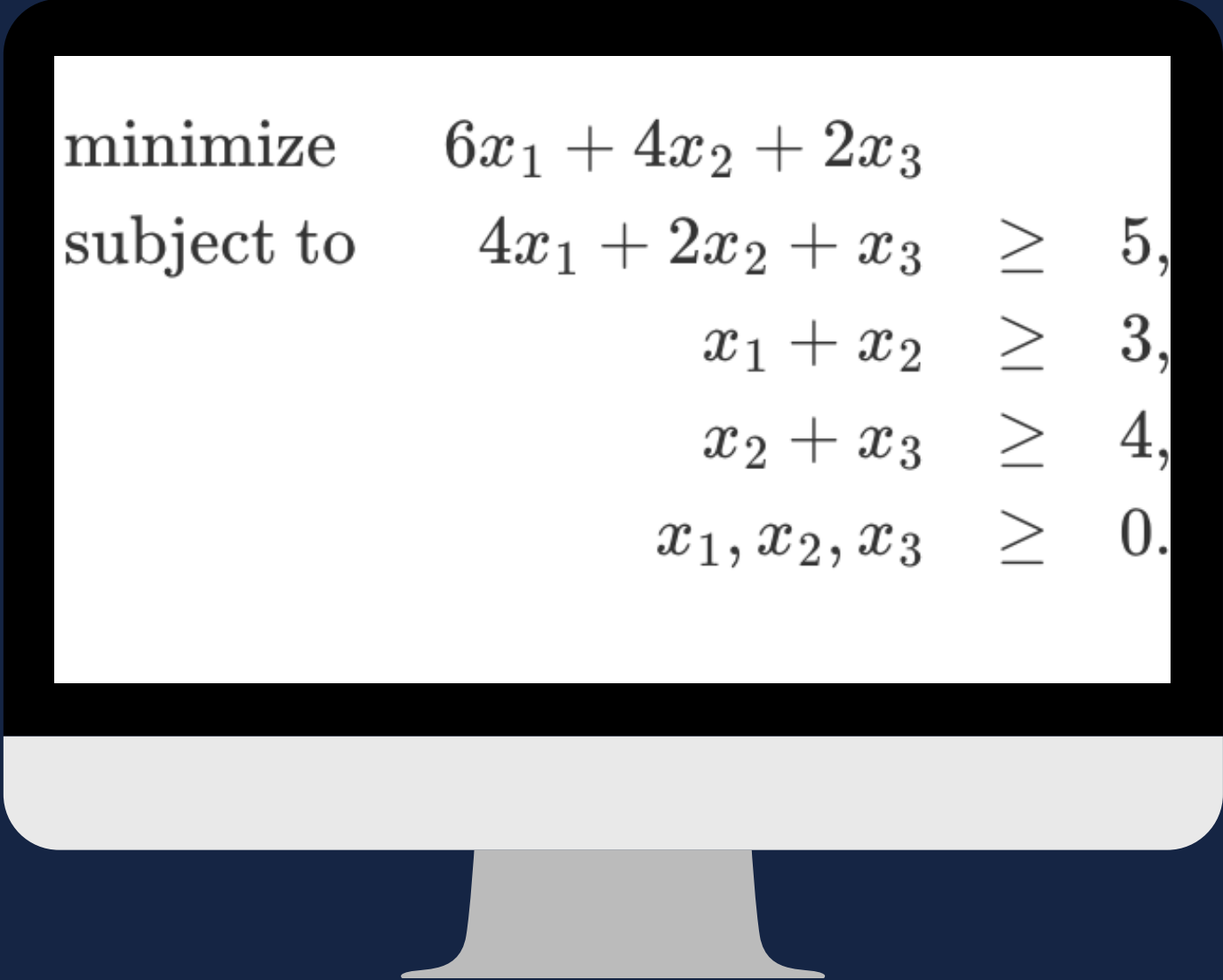




minimize  $6x_1 + 4x_2 + 2x_3$   
subject to  $4x_1 + 2x_2 + x_3 \geq 5,$   
 $x_1 + x_2 \geq 3,$   
 $x_2 + x_3 \geq 4,$   
 $x_1, x_2, x_3 \geq 0.$

## ① 변수 정의

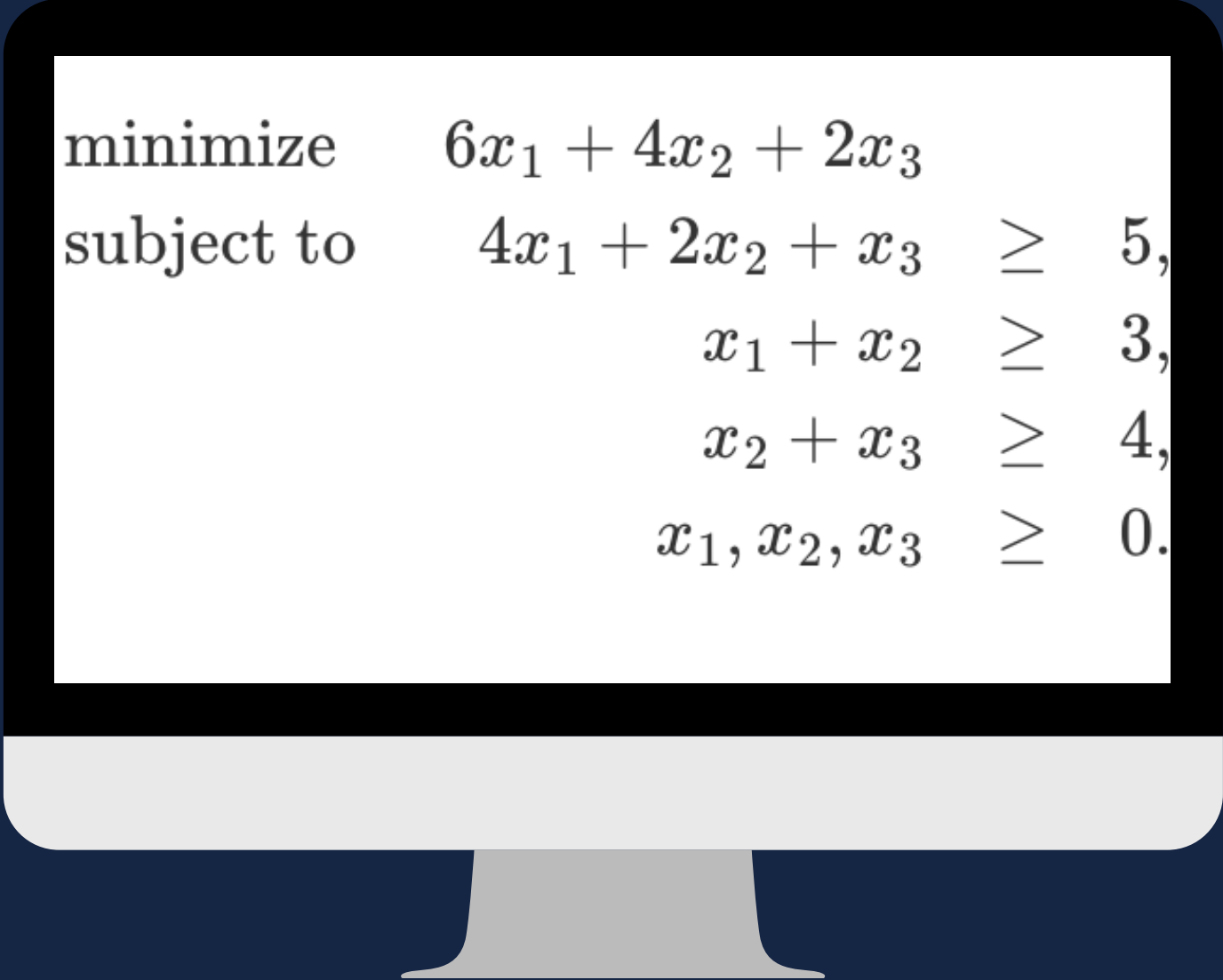
```
x1 = pulp.LpVariable('x1', lowBound = 0)  
x2 = pulp.LpVariable('x2', lowBound = 0)  
x3 = pulp.LpVariable('x3', lowBound = 0)
```



minimize  $6x_1 + 4x_2 + 2x_3$   
subject to  $4x_1 + 2x_2 + x_3 \geq 5,$   
 $x_1 + x_2 \geq 3,$   
 $x_2 + x_3 \geq 4,$   
 $x_1, x_2, x_3 \geq 0.$

## ② 문제 정의

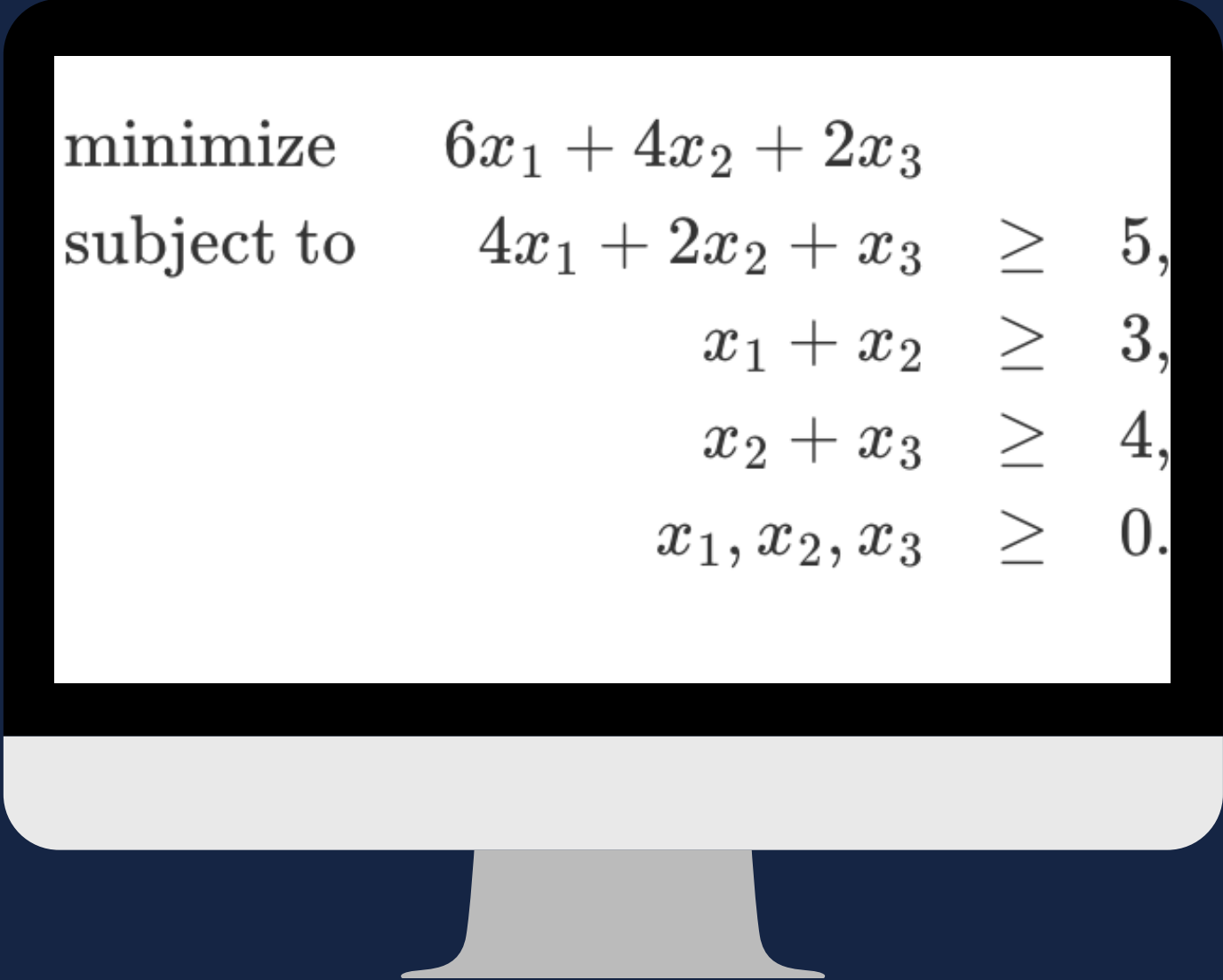
```
model = pulp.LpProblem("problem", pulp.LpMinimize)
```



minimize  $6x_1 + 4x_2 + 2x_3$   
subject to  $4x_1 + 2x_2 + x_3 \geq 5,$   
 $x_1 + x_2 \geq 3,$   
 $x_2 + x_3 \geq 4,$   
 $x_1, x_2, x_3 \geq 0.$

### ③ 목적함수 정의

```
model += 6 * x1 + 4 * x2 + 2 * x3
```



minimize  $6x_1 + 4x_2 + 2x_3$   
subject to  $4x_1 + 2x_2 + x_3 \geq 5,$   
 $x_1 + x_2 \geq 3,$   
 $x_2 + x_3 \geq 4,$   
 $x_1, x_2, x_3 \geq 0.$

#### ④ 제약조건 정의

```
model += 4 * x1 + 2 * x2 + x3 >= 5  
model += x1 + x2 >= 3  
model += x2 + x3 >= 4
```

```
# model
model = pulp.LpProblem('test_lp', pulp.LpMinimize)

# objective function
model += 6 * x1 + 4 * x2 + 2 * x3

# constraints
model += 4 * x1 + 2 * x2 + x3 >= 5
model += x1 + x2 >= 3
model += x2 + x3 >= 4

# solve
model.solve()

# optimal value
print(pulp.value(model.objective))

# optimal solution
print(x1.varValue)
print(x2.varValue)
print(x3.varValue)
```

## ⑤ 최적해 도출

`model.solve()`

-최적해를 담고있는 변수-

14.0  
0.0  
3.0  
1.0

`x1.varValue`

`x2.varValue`

`x3.varValue`

-목적함수의 최적 값-

`pulp.value(model.objective)`



# 이진 변수 설정 방법

```
X = pulp.LpVariable.dicts("x",lowBound=0,  
                           upBound=1, cat="Binary")
```





# 감사합니다!

2023 FIELD CAMP

