본 컨텐츠는 과학기술정보통신부 정보통신기획평가원의
SW중심대학지원사업단의 연구결과로 수행되었습니다.
(2019-0-01838)

과학기술정보통신부 IITP 정보통신기획평가원 배재대학교 PAI CHAI UNIVERSITY AI·SW중심대학사업단

파이썬 기초와 게임 만들기 - 3

Taejun Lee (marlrero@kakao.com, +82-10-5223-2912) Doctor's course
Paichai University, Dept. Computer Engineering, Lab. MIE

2023 동산고등학교-배재대학교 동아리 교육

CONTENTS

01

튜플과 레인지

1. 튜플
2. 튜플 연산
3. 튜플 vs. 리스트
4. 레인지
5. 레인지의 보폭
6. 타입 변환

1-1. 튜플

- 지금까지 우리가 배운 데이터 타입 정리
  - int형 데이터 예) 3, 5, 100
  - float형 데이터 예) 2.2 100.0 3.14159
  - 문자열 데이터 예) "I am a boy", "Girl", "Taejun"
  - 리스트 데이터 예) [1, 2, 3], ["AB", "CD], [2.5, "AB"]
  - 부울형 데이터 예) True, False
  - **튜플 데이터 예) (1, 2, 3), ("AB", "CD"), (2.5, "AB")**

1-1. 튜플

- 리스트와 비슷함
- 리스트는 값의 수정이 가능하다면, 튜플은 값의 수정이 불가능함

리스트는 [. . .]로 선언하고 튜플은 (. . .)로 선언한다.
튜플은 리스트와 마찬가지로 Python이 인식하는 데이터의 한 종류이다.
튜플은 리스트와 마찬가지로 하나 이상의 값을 묶는 용도로 사용한다.

```
>>> lst = [1, 2, 3]
>>> lst
[1, 2, 3]
>>> type(lst)
<class 'list'>
```

```
>>> tpl = (1, 2, 3)
>>> tpl
(1, 2, 3)
>>> type(tpl)
<class 'tuple'>
```

리스트는 추가/수정이 가능하다.

```
>>> lst = [1, 2, 3]
>>> lst.append(3)
>>> lst
[1, 2, 3, 3]
>>> lst[0] = 7
>>> lst
[7, 2, 3, 3]
```

1-1. 튜플

- 리스트는 값의 수정이 가능하다면, 튜플은 값의 수정이 불가능함

```
>>> tpl = (1, 2, 3)
>>> tpl
(1, 2, 3)
>>> tpl[0]
1
>>> tpl[0] = 7
Traceback (most recent call last):
  File “<pyshell#34>”, line 1, in <module>
    tpl[0] = 7
TypeError: 'tuple' object does not support item assignment
```

튜플은 추가/수정이 불가능하다.
즉, 처음 만들어진 그대로 사용해야 한다.
문자열(str)처럼 말이다.

```
>>> str = 'abc'
>>> str
'abc'
>>> str[0] = 'z'
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    str[0] = 'z'
TypeError: 'str' object does not support item assignment
```

1-2. 튜플 연산

리스트, 문자열 관련 함수에서 배웠죠?
튜플에서도 동작합니다.

- 리스트와 문자열과 유사함

```
>>> nums = (3, 2, 5, 7, 1)
>>> len(nums)                # 값의 개수는?
5
>>> max(nums)                # 최댓값은?
7
>>> min(nums)                # 최솟값은?
1
```

```
>>> nums = (1, 2, 3, 1, 2)
>>> nums.count(2)                  # 2가 몇 번 등장해?
2
>>> nums.index(1)          # 가장 앞에(왼쪽에) 저장된 1의 인덱스 값은?
0
```

```
>>> for i in (1, 3, 5, 7, 9):
        print(i, end = ' ')

1 3 5 7 9
```

튜플 기반으로 for 루프를 돌릴 수 있음

1-3. 튜플 vs. 리스트

- 값의 변경이 불가능한 튜플을 쓸 바엔,
  차라리 리스트를 쓰는 것이 더 편하지 않을까??

단순히 인덱스 숫자로 진우 데이터에 접근하고 있다.
만약, 0-1, 2-3, 4-5 짝이라고 생각하지 못하면
1과 2를 진우의 데이터라고 착각할 수도 있다.

```
>>> frns = ['동수', 131120, '진우', 130312, '선영', 130904]
>>> frns[2]  # 진우 이름 따로
'진우'
>>> frns[3]  # 진우 생년월일 따로
130312
```

그래서 리스트로 한 번 더 묶었다. (2차원, 표가 됨)

```
>>> frns = [['동수', 131120], ['진우', 130312], ['선영', 130904]]
>>> frns[1]
['진우', 130312]
```

만약, 진우의 생년월일 정보가 누군가의 실수로 바뀐다면?

그래서 튜플로 묶어서 안정성을 부여했다. 정보가 바뀌는 일은 없다.

```
>>> frns = [('동수', 131120), ('진우', 130312), ('선영', 130904)]
```

1-3. 튜플 vs. 리스트

- 2차원(표, 행렬): 튜플 안에 튜플, 리스트 안에 리스트

인덱스를 이용해 2번 접근한다.

```
>>> frns = [['동수', 131120], ['진우', 130312], ['선영', 130904]]
>>> frns[1][1] = 130102
>>> frns
[['동수', 131120], ['수진', 130102], ['선영', 130904]]
```

```
>>> frns = [('동수', 131120), ('진우', 130312), ('선영', 130904)]
>>> frns[0][0]
'동수'
>>> frns[0][1]
131120
```

1-4. 레인지

- 범위를 지정할 때 사용

[복습] 레인지 함수는 for 루프에 많이 사용된다.

```
>>> for i in range(1, 11):
        print(i, end = ' ')

1 2 3 4 5 6 7 8 9 10
```

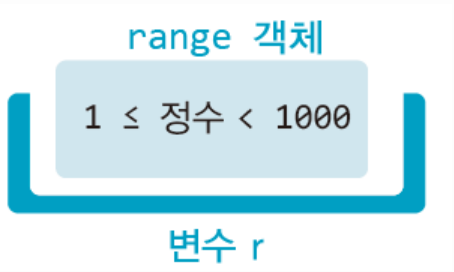
➢ 레인지 함수는 리스트나 튜플이 아닌, 그 자체로 레인지임

```
>>> r = range(1, 10)
>>> type(r)
<class 'range'>
```

컴퓨터공학에서는 시작은 0부터,
범위 지정 시 반열림구간을 사용하는 것이 보편적입니다!

➢ 시작 숫자 ~ 끝 숫자 – 1 → 반열림구간

```
>>> r = range(1, 1000)
```



1-4. 레인지

- 레인지와 in 연산자

```
>>> r = range(1, 10)
>>> 9 in r
True
>>> 10 not in r
True
```

이렇듯, 레인지는 시작 숫자 ~ 끝 숫자 - 1 임을 알 수 있다.

연습 문제

- [예제 1-2] 아래와 같은 리스트를 선언한다고 가정한다.
  (행은 학생의 성적, 각 열은 국어 점수, 수학 점수, 영어 점수이다.)

```
score = [
        [100, 50, 20],
        [75, 40, 30],
        [95, 100, 89],
        [67, 85, 70]
]
```

- 예제 1. 4명 학생의 평균을 각각 구하라.
- 예제 2. 학생 전체를 대상으로 국어 점수, 수학 점수, 영어 점수의 평균을 각각 구하라.

1-5. 레인지의 보폭

- 보폭(steps)의 기본 값은 1이고, 보폭을 주면 일정하게 띄어주는 역할

```
>>> range(1, 10, 2)          # 1부터 10 이전까지 2씩 증가하는 레인지
range(1, 10, 2)
>>> range(1, 10, 3)          # 1부터 10 이전까지 3씩 증가하는 레인지
range(1, 10, 3)
```

```
>>> list(range(1, 10, 2))  # 1부터 10 이전까지 2씩 증가하는 리스트 만들기
[1, 3, 5, 7, 9]
>>> list(range(1, 10, 3))  # 1부터 10 이전까지 3씩 증가하는 리스트 만들기
[1, 4, 7]
```

```
>>> list(range(10, 2))
[]
>>> list(range(10, 2, 1))
[]
```

시작 숫자가 끝 숫자보다 크다고 거꾸로 된 레인지를 만들어주지 않는다.
시작 숫자가 끝 숫자보다 크면,
step을 음수로 줘야 거꾸로 된 레인지 생성이 가능하다.

```
>>> list(range(10, 2, -1))   # 10부터 1씩 감소하여 3까지 이르는 정수들
[10, 9, 8, 7, 6, 5, 4, 3]
>>> list(range(10, 2, -2))   # 10부터 2씩 감소하여 3까지 이르는 정수들
[10, 8, 6, 4]
>>> list(range(10, 2, -3))   # 10부터 3씩 감소하여 3까지 이르는 정수들
[10, 7, 4]
```

1-6. 타입 변환

- 리스트 함수로 튜플, 레인지, 문자열 변환

```
>>> list((1, 2, 3))                    # 튜플을 리스트로
[1, 2, 3]
>>> list(range(1, 5))       # 레인지를 리스트로
[1, 2, 3, 4]
>>> list("Hello")                              # 문자열을 리스트로
['H', 'e', 'l', 'l', 'o']
```

- 튜플 함수로 리스트, 레인지, 문자열 변환

```
>>> tuple([1, 2, 3])                   # 리스트를 튜플로
(1, 2, 3)
>>> tuple(range(1, 5))      # 레인지를 튜플로
(1, 2, 3, 4)
>>> tuple("Hello")                     # 문자열을 튜플로
('H', 'e', 'l', 'l', 'o')
```

1-6. 타입 변환

- 레인지를 리스트나 튜플로 변환

```
>>> lst = list(range(1, 16))
>>> lst
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
>>> tpl = tuple(range(1, 16))
>>> tpl
(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15)
```

연습 문제

- 예제 3. 구구단의 7단을 거꾸로 출력하는 코드를
  for 루프와 range를 기반으로 만들어라.
  단, 출력은 다음과 같이 결과를 거꾸로 보여라.
  예) 63 56 49 42 35 28 21 14 7

- 예제 4. 튜플을 1부터 시작해서 100까지 증가하도록 만들고,
  그리고 다시 1씩 줄어들어서
  마지막에 1로 끝나도록 만들어라.
  예) (1, 2, 3, ..., 97, 98, 99, 100, 99, 98, 97, ..., 5, 4, 3, 2, 1)
  물론 위의 숫자를 모두 입력해서
  만들라는 의미가 아니고
  레인지와 이를 튜플로 바꿔주는
  함수를 사용해 한 줄에 입력 가능한 수준으로 만들어라.
  힌트) 값이 증가하는 튜플과
  감소하는 튜플을 각각 생성해 이를 하나로 묶기

02

함수

1. 정의와 호출
2. 함수의 입력 I
3. 함수의 출력
4. 함수의 중요성
5. 함수의 입력 II

2-1. 정의와 호출

- 우리가 사용한 함수들:
  print(), input(), type(), int(), float(), str(), list(), len(), ...

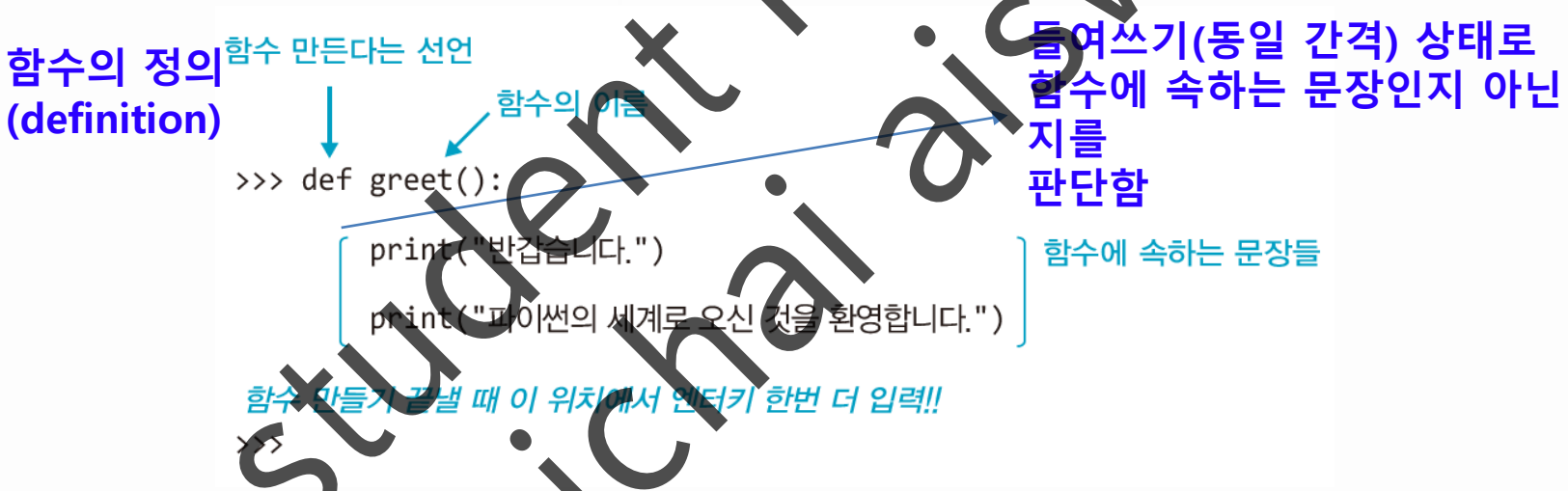
```
>>> num = float("3.14")
>>> type(num)
<class 'float'>
```

```
>>> height = int(input("키 정보 cm 단위로 입력: "))
키 정보 cm 단위로 입력: 178
>>> print(height)
178
```

```
>>> st = [1, 2, 3]
>>> st.remove(2)          # 리스트에서 2를 찾아서 삭제
>>> st
[1, 3]
```

2-1. 정의와 호출

- 함수(function): 코드를 하나의 기능으로 묶어서
  나중에 재사용하기 편하도록 하기 위함
  - 수학에서 말하는 함수와 비슷함
  - 입력과 출력이 존재함
  - 함수를 만들 때(정의할 때)는 "def"라는 키워드를 사용함



2-1. 정의와 호출

- 함수를 만들었으면, 사용(호출, call, invoke)해야 함 (사용은 여러 번 가능)
- 함수 안에 있는 문장이 모두 실행이 완료되면,
  원래 사용했던 것으로 되돌아옴

함수의 정의
(definition)

함수의 호출
(call, invoke)

```
>>> def greet():
        print("반갑습니다.")
        print("파이썬의 세계로 오신 것을 환영합니다.")

>>> greet()
반갑습니다.
파이썬의 세계로 오신 것을 환영합니다.
>>> greet()
반갑습니다.
파이썬의 세계로 오신 것을 환영합니다.
```

2-2. 함수의 입력 I

- 수학에서 함수와 마찬가지로 입력과 출력이 존재함
- 컴퓨터공학에서 함수의 입력의 용어는 아래와 같음 (이름을 알 필요는 없음)
  - 매개변수(parameter): 함수를 사용할 때 받을 값을 저장하기 위한 변수
  - 인자(argument): 함수를 사용할 때 함수에게 전달할 값

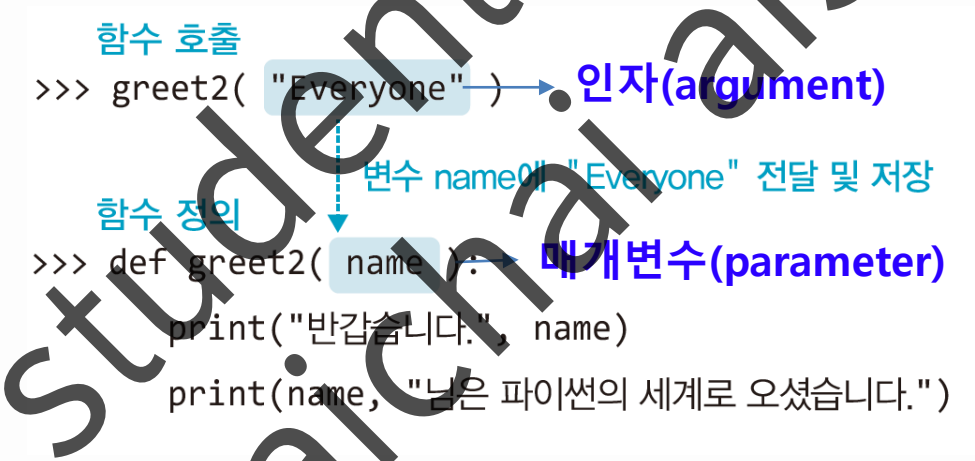
```
>>> def greet2(name):
        print("반갑습니다.", name)
        print(name, "님은 파이썬의 세계로 오셨습니다.")

>>> greet2("John")
반갑습니다. John
John 님은 파이썬의 세계로 오셨습니다.
>>> greet2("Yoon")
반갑습니다. Yoon
Yoon 님은 파이썬의 세계로 오셨습니다.
```

매개변수(parameter)

인자(argument)

2-2. 함수의 입력 I

- 수학에서 함수와 마찬가지로 입력과 출력이 존재함
- 컴퓨터공학에서 함수의 입력의 용어는 아래와 같음 (이름을 알 필요는 없음)
  - 매개변수(parameter): 함수를 사용할 때 받을 값을 저장하기 위한 변수
  - 인자(argument): 함수를 사용할 때 함수에게 전달할 값
  - 함수의 입력은 1개 만이 아니라 여러 개도 가능
    → 매개변수와 인자 사이에 쉼표(,)로 구분함



2-3. 함수의 출력

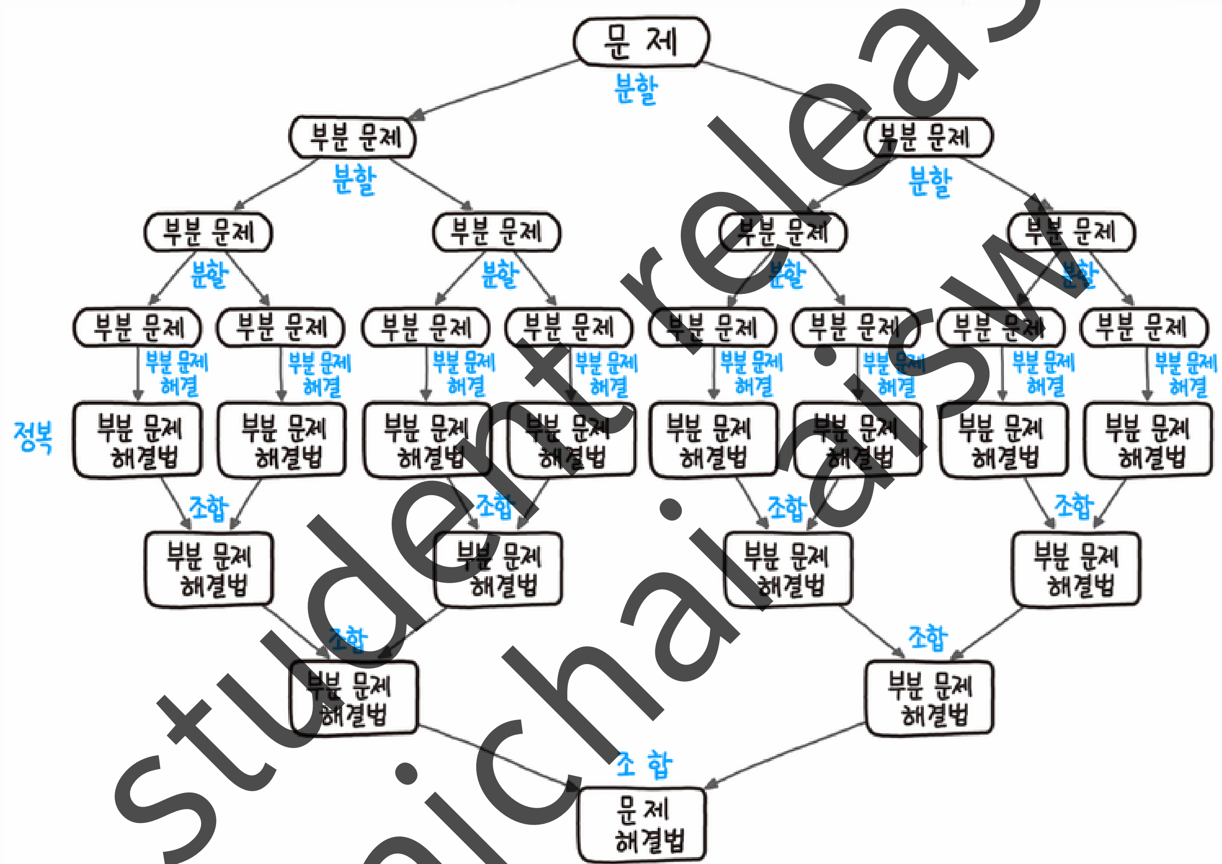
- 함수의 출력을 컴퓨터공학에서는 주로 반환(return)이라고 부름
  - 반환의 2가지 의미:
    함수를 종료한다 + return 오른쪽에 있는 값을 원래 부른 곳에 돌려줌
  - 함수의 종료: 원래 부른 곳(호출한 곳)으로 돌아감(return)

```
>>> def adder2(num1, num2):
        ar = num1 + num2
        return ar

>>> result = adder2(5, 3)
>>> print(result)
8
```

```
>>> def adder3(num1, num2):
        return num1 + num2

>>> print(adder3(5, 3))
8
```

2-4. 함수의 중요성

- 큰 프로그램을 만들기 위해 작은 단위로 쪼개서 해결(분할 정복)하는 것이 좋음



파이썬 알고리즘 인터뷰 2020

2-4. 함수의 중요성

- 프로그래머는 똑같은 코드의 반복으로 소스 코드가 길어지는 것을 싫어함
  예) 깜지 쓰는 거 좋으신가요?
  예) 계속 똑같은 반복 일상이 좋으신가요?

- 유능한 개발자가 만들어 놓은 함수를 사용하면
  프로그램을 작성하는 시간이 매우 단축됨
  예) print(), input(), type() 등 여러분들이 파이썬에서 사용했던 함수들

- 한 번 함수를 만들면 다른 프로그램에 어식하지도 편리함
  예) 내가 원의 넓이를 구하는 함수를 A 프로그램에서
  만들어 사용했다면,
  B 프로그램에서 이를 따로 구현하지 않고 바로 사용 가능

- 어떤 지점의 버그를 찾기 편리함 → 함수의 문제가 있다면 거기만 고침!

연습 문제

- 예제 5. 하나의 정수를 입력 받아 부호를 반대로 출력하는 inverse 함수를 정의하라.

- 예제 6. 두 개의 정수를 입력받아 평균값을 계산하여 출력하는 average 함수를 정의하라.

- 예제 7. 홀수인지 짝수인지를 판단하는 is_odd_even 함수를 정의하라.

- 예제 8. 구구단에서 단을 입력 받아 해당하는 단을 출력하는 gugudan 함수를 정의하라.

2-5. 함수의 입력 II

- 함수한테 값(인자)을 전달할 때 이름을 명시해서 줄 수도 있음

```
>>> for i in (1, 3, 5, 7, 9):
        print(i, end = ' ')

1 3 5 7 9
```

```
>>> print(1, 2, 3, 4, sep = '#')

1#2#3#4
```

매개변수의 순서가 바뀌어도 이름을 지정했으므로 상관없음

```
>>> def who_are_you(name, age):
        print("이름:", name)
        print("나이:", age)
```

```
>>> who_are_you(name = "이태준", age = 25)
이름: 이태준
나이: 25
```

```
>>> who_are_you(age = 25, name = "이태준")
이름: 이태준
나이: 25
```

2-5. 함수의 입력 II

- 함수를 만들 때(정의할 때) 입력(매개변수)의 기본값을 지정할 수 있음

```
>>> def who_are_you(name, age = 0):    # age의 디폴트 값은 0
        print("이름:", name)
        print("나이:", age)
```

age를 채울 값이 전달되지 않으면(즉, 사용자가 이를 빠뜨리면) 0을 대신 전달해주겠다!

```
>>> who_are_you("줴임스~")
이름: 줴임스~
나이: 0
>>> who_are_you("쟌~", 29)
이름: 쟌~
나이: 29
```

- 주의사항: 기본값을 줄 때는 뒤에서부터 와야 함

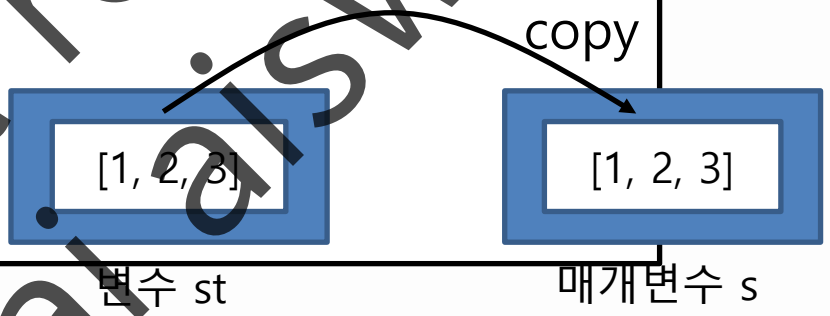
```
>>> def who_are_you(age = 0, name):    # 기본 값의 위치 오류 발생
        print("이름:", name)
        print("나이:", age)
```

반드시 기본 값을 갖는 매개변수는 뒤에 와야 한다. Why?

2-5. 함수의 입력 II

- 함수의 입력으로는 우리가 배웠던 타입들이 올 수 있음
  → 주의: 만들어 놓은(정의한) 함수에게 값(인자)을 전달할 때
  매개변수로 값이 복사되는 것이 아니고 참조된다(가리킨다).

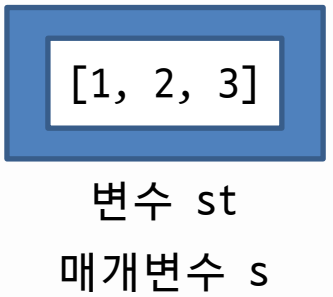
```
>>> def func(s):    # 전달되는 값을 리스트라고 가정
        s[0] = 0    # 리스트의 첫번째 값을 0으로 수정
        s[-1] = 0   # 리스트의 마지막 값을 0으로 수정

>>> st = [1, 2, 3]
>>> func(st)
>>> st
[0, 2, 0]
```



st에 담겨 있는 리스트 [1, 2, 3]이 매개변수 s에 어떻게 전달되었을까?

이렇게 복사가 될까?

다음과 같이 메모리 공간에 하나의 이름을 더 붙이는 방식으로 처리함



연습 문제

- 예제 9. 다음과 같이 출력하도록
  for 루프 안에 코드를 채워라. (안에 코드는 여러 줄임)
  출력 예) 1 2 3
  2 3 4
  3 4 5
  코드) for i in range(3):
  _____

- 예제 10. 리스트를 함수의 입력으로 하여
  아래의 코드가 동작할 수 있도록 하라.

```
>>> def add_list(s):
        _____________  # add_list 함수 정의, 여러 줄에 걸쳐서 만듦
>>> st = [1, 2, 3, 4, 5]
>>> add_list(st)
>>> st
[2, 3, 4, 5, 6]
```

03 모듈

1. 통합 개발 환경
2. Visual Studio Code
3. 모듈 만들기
4. 모듈 불러오기
5. 빌트인 모듈과 함수

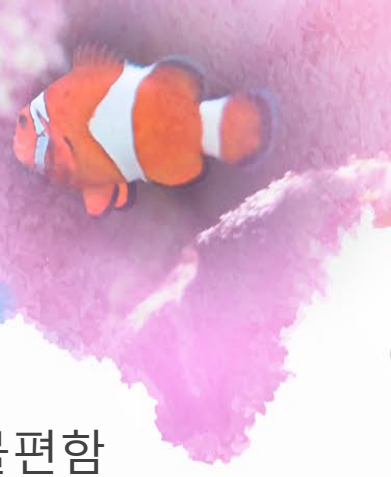
3-1. 통합 개발 환경

- 통합개발환경(IDE: Integrated Development Environment)
  - 소프트웨어(프로그램) 개발을 지원하는 도구
    → 소스 코드 작성 + 디버깅(debugging) + 프로그램에 필요한 데이터 관리 등
  - 버그(bug): 프로그램 실행 시 발생하는 에러
    → 사용자가 발견할 수 있고, 개발자가 발견할 수도 있음
    → 이 에러를 고치는 행위를 디버깅이라고함

스미소니언 박물관에 보관중인
위 사진은 최초의 버그.
컴퓨터 회로 안에 벌레가 들어가
합선된 것으로 추정.

  - 통합 개발 환경은 다양하게 있음
    - 파이썬 설치 시 기본 제공하는 IDLE
    - PyCharm
    - **Visual Studio Code** 등...
  - 통합 개발 환경은 협업 시 고려해야 할 사항
    → 협업하지 않고 혼자 개발한다면 자신이 편한 것을 쓰면 됨

3-1. 통합 개발 환경



- Python IDLE의 특징
  - 셸(shell) 환경으로 질의응답 방식으로 코드 작성이 편함
    → 코드가 길어질 경우에는 불편함이 존재
  - 앞으로 배울 모듈(module) 방식으로 코딩하고자 할 때, 모듈 관리에서 불편함
  - 다른 통합 개발 환경에 비해 편리하지 않음
    예) 실행을 즉각적으로 확인할 수 있는 Jupyter notebook의 예
    → 데이터 분석에서 그래프나 중간 결과 확인용으로 많이 사용

3-1. 통합 개발 환경

jupyter spectrogram (autosaved)

File Edit View Insert Cell Kernel Help Python 3

Markdown CellToolbar

Simple spectral analysis

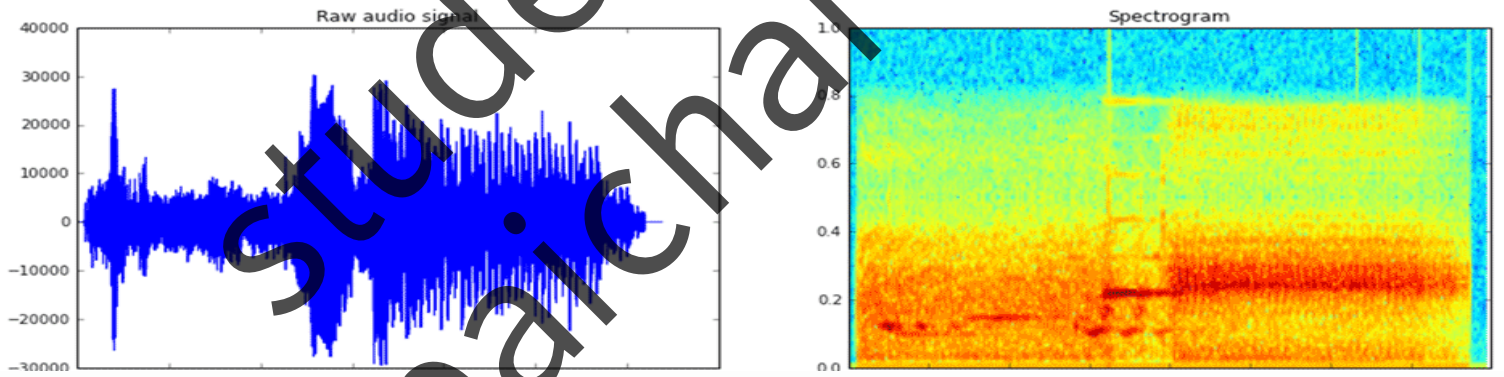
An illustration of the Discrete Fourier Transform

$$X_k = \sum_{n=0}^{N-1} x_n exp^{\frac{-2\pi i}{N} kn} \quad k = 0, \ldots, N-1$$

```
In [2]: from scipy.io import wavfile
        rate, x = wavfile.read('test_mono.wav')
```

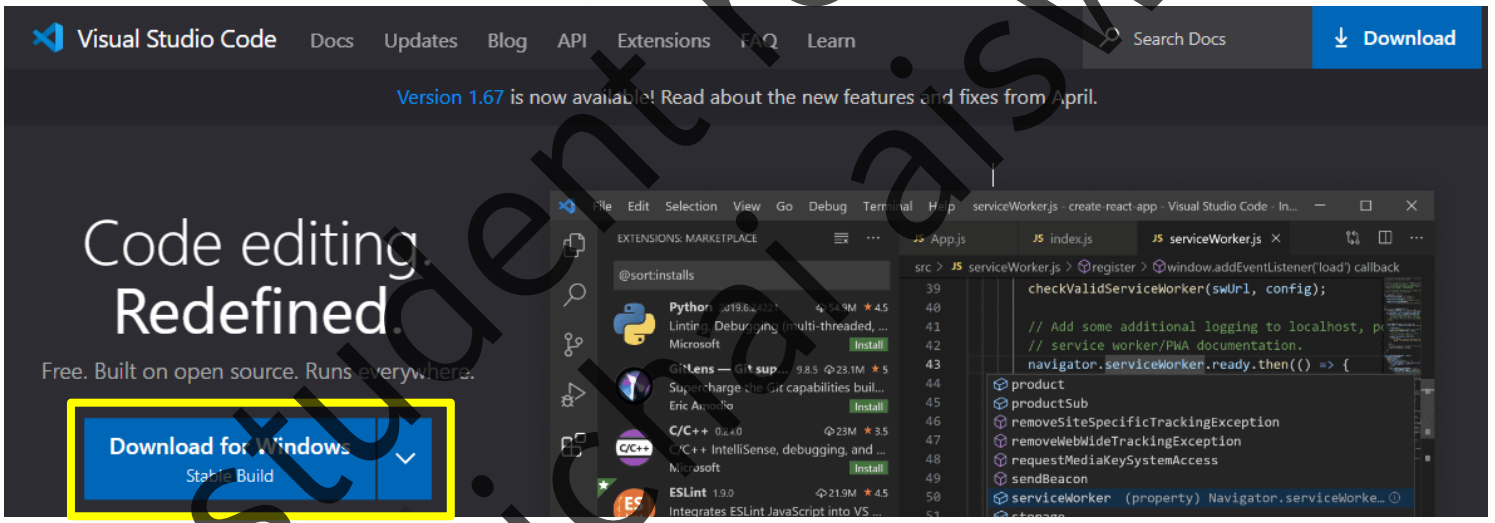
And we can easily view it's spectral structure using matplotlib's builtin specgram routine:

```
In [5]: fig, (ax1, ax2) = plt.subplots(1,2,figsize(16,5))
        ax1.plot(x); ax1.set_title('Raw audio signal')
        ax2.specgram(x); ax2.set_title('Spectrogram');
```



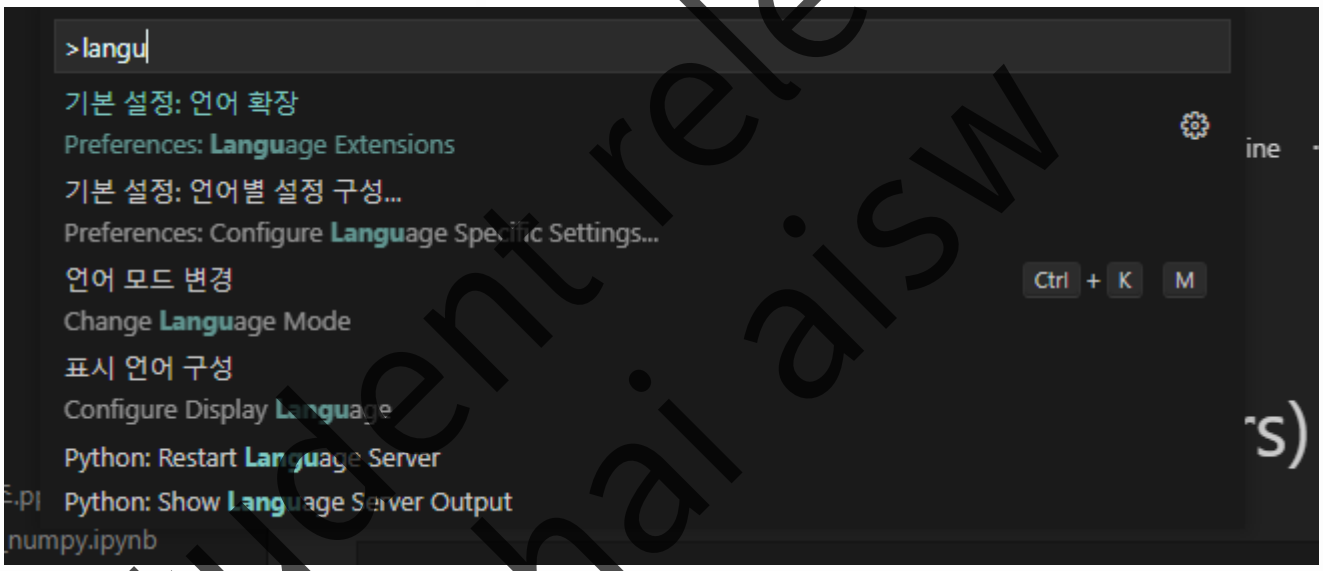
3-2. Visual Studio Code

- Microsoft에서 제작한 오픈소스 통합개발환경
- Python 뿐만 아니라, C/C++, Java, JavaScript 등 다양한 환경 지원
- Linux의 Ubuntu의 기본 텍스트 에디터 채택
- 설치 링크: https://code.visualstudio.com/



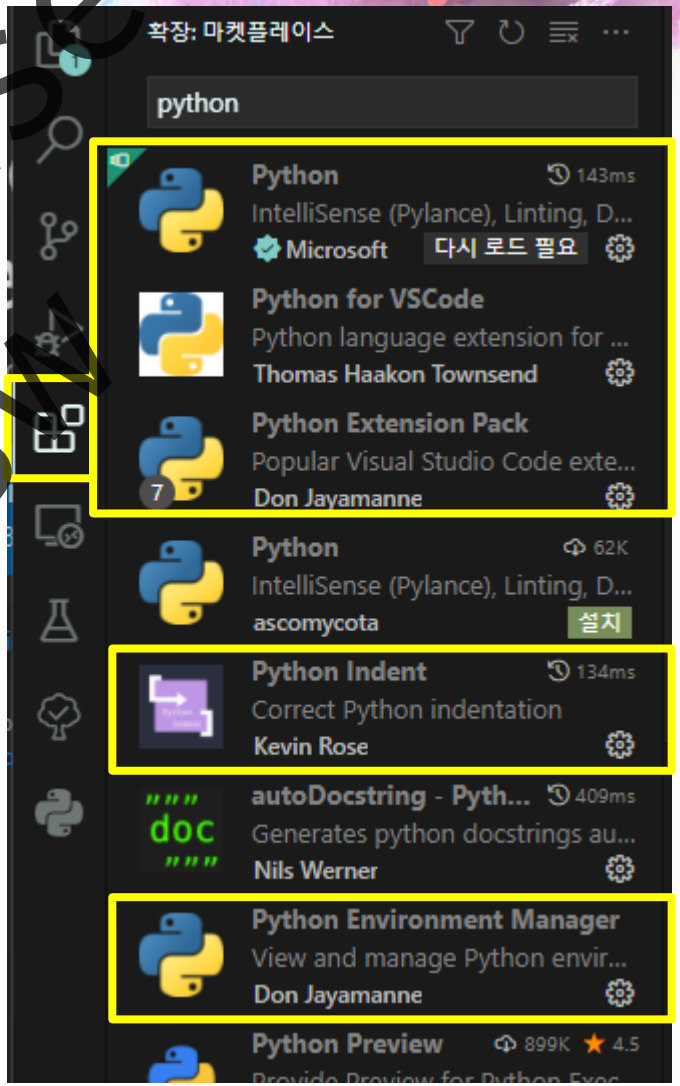
3-2. Visual Studio Code

- 초기 설정
  - Ctrl + Shift + P → "language" 입력 → 한글 언어 팩 다운로드 후 재시작



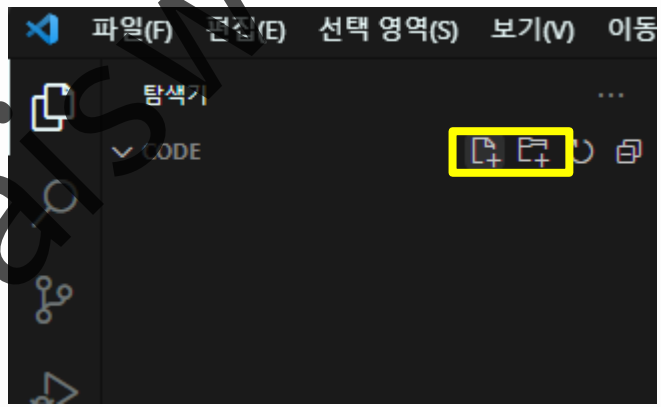
3-2. Visual Studio Code

- 초기 설정
  - Python 환경 설정을 위해 확장 프로그램에서 "python" 입력 후 설치



3-2. Visual Studio Code

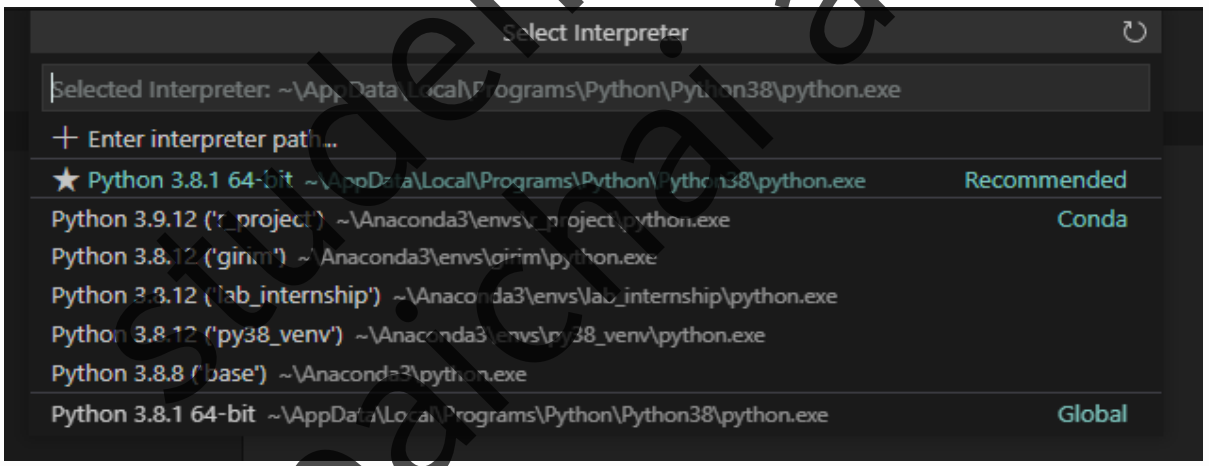
- 작업 폴더를 열고나서 circle.py 파일 만들기 (폴더를 만들어도 됨)



3-2. Visual Studio Code

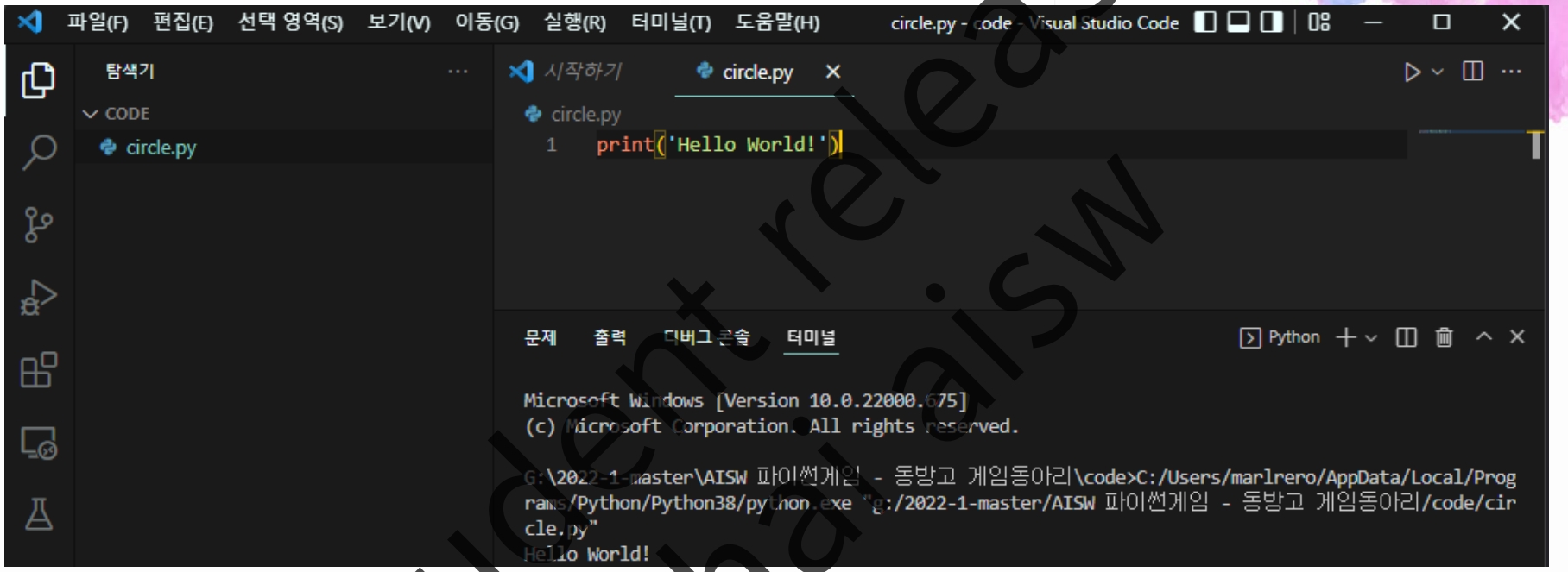
- Python Interpreter(해석기) 설정
  → 한 컴퓨터에는 Python Interpreter가 여러 개 있을 수 있음
  (왜? 버전이 다를 수도 있기 때문)
  - Ctrl + Shift + P → "python: select" 입력

  - 여러분이 설치한 Python Interpreter에 맞게 선택



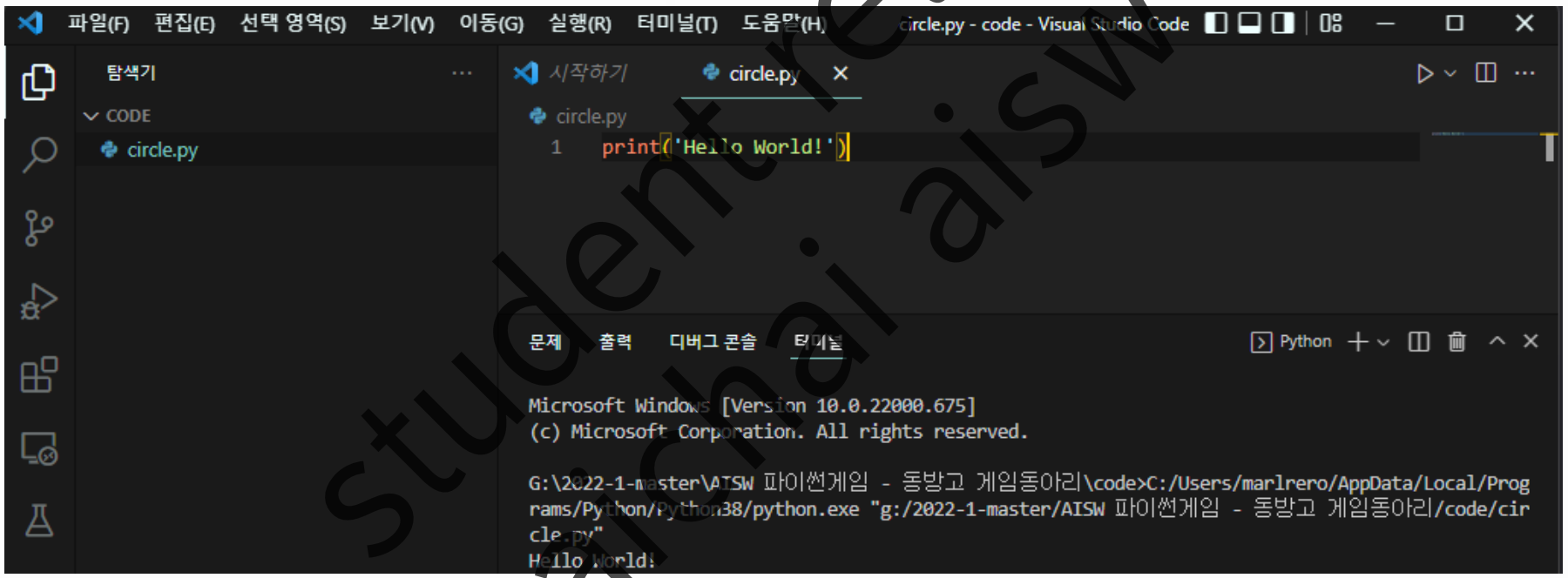
3-2. Visual Studio Code

- "Hello World!" 출력을 통해 제대로 작동하는지 확인



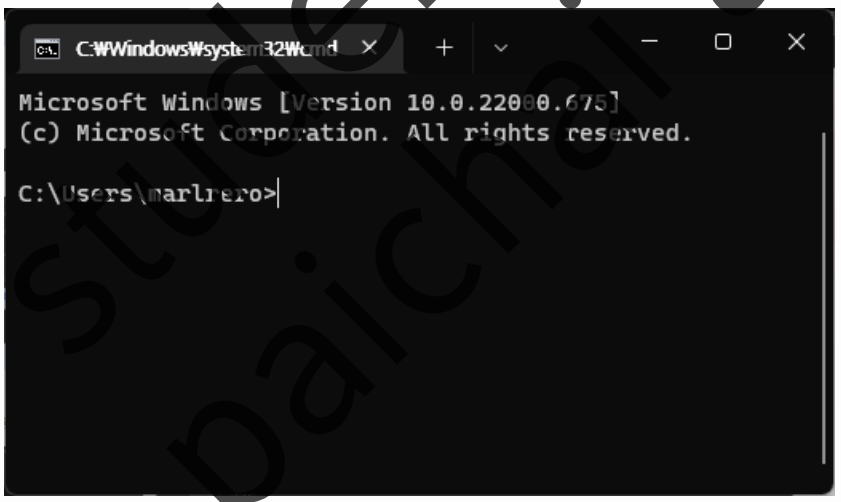
3-2. Visual Studio Code

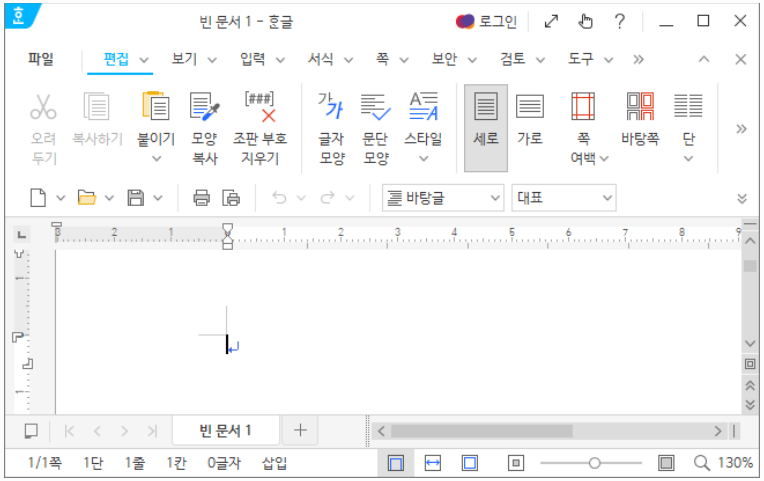
- 화면 구성
  - 왼쪽: 탐색기(폴더/파일 보기), 검색, 확장 프로그램, 디버깅 등
  - 오른쪽: 소스코드 작성 (자동완성 및 도움말 제공)
  - 오른쪽 아래: 실행하면 결과 출력 (터미널에서 출력)



3-2. Visual Studio Code

- 터미널(Terminal): VS Code의 화면 구성 오른쪽 아래의 정체
  - 콘솔(console)이라고도 부름
  - 컴퓨터에게 명령어(command)를 입력해 결과를 텍스트 형태로 받을 수 있는 형태
  - 최근에는 GUI(Graphic User Interface) 환경이라 잘 사용하지 않음 → 개발자의 경우 명령어 형태가 편리해 많이 선호함
  - 여러분들이 컴퓨터/해커 관련 영화/드라마에서 많이 보던 그것...
  - Windows 환경: 윈도우 키 + R → "cmd" 입력
  - 몇 개의 명령어?
    - dir
    - cd
    - ipconfig
    - ...





3-3. 모듈 만들기

- circle.py 라는 이름의 파일에 아래와 같이 원 둘레와 넓이를 구하는 함수를 넣고 저장

```
# circle.py
PI = 3.14                          # 원주율

def ar_circle(rad):                  # 원의 넓이를 계산해서 반환하는 함수
        return rad * rad * PI

def ci_circle(rad):                # 원의 둘레를 계산해서 반환하는 함수
        return rad * 2 * PI
```

- 모듈(module): 이렇게 만들어진 하나의 소스 파일을 의미
  - 가져다 쓸 수 있는, 다른 프로그램의 일부가 될 수 있는 내용을 담고 있는 파일
  - 보편적으로 Python에서는 소스 파일을 가리켜 '모듈'이라고 함

3-4. 모듈 불러오기

- import: 모듈을 불러옴

circle.py와 circle_test1.py는 같은 폴더에 위치해야 함

```
# circle_test1.py
import circle      # circle.py 모듈을 가져다 쓰겠다는 선언!

def main():
    r = float(input("반지름 입력: "))
    ar = circle.ar_circle(r)   # circle.py의 ar_circle 함수 호출
    print("넓이:", ar)
    ci = circle.ci_circle(r)   # circle.py의 ci_circle 함수 호출
    print("둘레:", ci)

main()
```

파일의 확장자 .py를 쓰지 않음

```
반지름 입력: 5.5
넓이: 94.985
둘레: 34.54
```

3-4. 모듈 불러오기

- from 모듈 import 모듈 안의 함수(or 변수 or 클래스 ...)

circle.py와 circle_test1.py는 같은 폴더에 위치해야 함

```
# circle_test2.py
from circle import ar_circle
from circle import ci_circle

def main():
        r = float(input("반지름 입력: "))
        ar = ar_circle(r)                    # circle.py의 ar_circle 함수 호출
        print("넓이:", ar)
        ci = ci_circle(r)                    # circle.py의 ci_circle 함수 호출
        print("둘레:", ci)

main()
```

from 모듈 import 함수/클래스 이름

```
from circle import ar_circle, ci_circle
```

이렇게 하면 모듈 이름 없이 함수 이름만으로 호출 가능

```
반지름 입력: 5.5
넓이: 94.985
둘레: 34.54
```

3-4. 모듈 불러오기

- 모듈의 함수 이름과 내가 만든 함수의 이름이 우연히 중복되는 경우

```
# circle_simple2.py
from circle import ci_circle

def ci_circle(rad):
    print("둘레: ", rad * 2 * 3.14)

def main():
        r = float(input("반지름 입력: "))
        ci_circle(r)     # ? ? ?
        ci_circle(r)     # ? ? ?
    . . .

main()
```

```
# circle.py
. . . .
def ci_circle(rad):
        return rad * 2 * PI
```

Python: ci_circle 함수가 너가 만든 거야?
아니면 circle 모듈의 ci_circle 이야?

3-4. 모듈 불러오기

- 모듈의 함수 이름과 내가 만든 함수의 이름이 우연히 중복되는 경우

as로 모듈의 클래스나 함수의 별명을 지을 수 있다.
그러면 구분이 가능하다.

```
# circle_simple2.py
from circle import ci_circle as cc

def ci_circle(rad):
    print("둘레: ", rad * 2 * 3.14)

def main():
        r = float(input("반지름 입력: "))
        ci_circle(r)    # circle_simple2.py의 ci_circle 함수
        cc(r)           # circle.py의 ci_circle 함수
    . . .

main()
```

```
# circle.py
. . . .
def ci_circle(rad):
        return rad * 2 * PI
```

3-4. 모듈 불러오기

- as: 모듈 이름을 줄이는 별명의 역할

바로 이전에서는 as로 모듈의 함수의 이름을 임시로 바꿨다(별명을 지어줬다).

이렇게 모듈 자체에 대해 이름을 임시로 바꿀 수 있다(별명을 지어줄 수 있다).

```
# circle_test3.py
import circle as cc
def main():
    r = float(input("반지름 입력: "))
    ar = cc.ar_circle(r)
    . . . .
```

3-5. 빌트인 모듈과 함수

- 우리가 썼던 print(), int() 이런 것은 어떤 모듈인가요?
  - 우리가 만들지 않았던 함수의 정체는?
  - 빌트인(built-in) 가전제품:
    처음 집이 지어질 때 함께 세팅되어 들어가는 가전제품


LG전자

빌트인 냉장고


LG전자

빌트인 세탁기


LG전자

빌트인 오븐

  - 빌트인 함수(built-in function): Python을 설치하면 기본으로 제공되는 함수

```
>>> print
<built-in function print>
>>> input
<built-in function input>
```

이렇게 함수의 이름만 입력하면
프롬프트가 이렇게 답변한다.

3-5. 빌트인 모듈과 함수

- 빌트인 모듈(built-in module): Python 기본 제공 모듈
  - 모듈이 어디에 저장되어 있는지 신경 쓸 필요가 없음
  - 빌트인 모듈과 함수를 모두 외울 필요 없음 (많이 사용하면 익숙해짐)
  - 언제든지 import 선언으로 그 안에 있는 기능 사용 가능
  - 참고: https://docs.python.org/3.8/library/index.html
- 다른 사람이 만든 모듈을 가져다 쓸 수도 있음
  (pip(preferred installer program), 다음 수업 때...)

```
>>> import math
>>> math.fabs(-10)
10.0
```

```
math.sin(x)          sin x
math.cos(x)          cos x
math.tan(x)          tan x
math.asin(x)         arcsin x
math.acos(x)         arccos x
math.atan(x)         arctan x
. . . .
```

04

딕셔너리

1. 정의
2. 딕셔너리 연산
3. 딕셔너리와 for문

4-1. 정의

- 사전(dictionary)과 유사한 데이터 종류
- 중괄호({ … })로 감싸서 표현
- 중괄호 안에는 key: value 형태
- 키(key)와 값(value)는 서로 쌍을 이룸
- 키와 값은 무엇이든지 될 수 있음 (단, 리스트는 키로 올 수 없음)

```
>>> dc = {
        '코카콜라': 900,
        '바나나맛우유': 750,
        '비타500': 600,
        '삼다수': 450
        }
>>> dc
{'코카콜라': 900, '바나나맛우유': 750, '비타500': 600, '삼다수': 450}
```

4-1. 정의

- 딕셔너리에 저장되는 값의 종류는 각각 달라도 됨

```
>>> dc = {
          '이름': '이순둥',  # 값이 문자열
          '나이': 19,        # 값이 정수
          '직업': '학생',    # 값이 문자열
          '키': 175.8        # 값이 실수
          }
>>> dc
{'이름': '이순둥', '나이': 19, '직업': '학생', '키': 175.8}
```

- 값은 중복 가능하나, 키는 "값을 꺼내는 열쇠"이므로 중복 불가

```
>>> dc = {
          '이순둥': 22,
          '정순둥': 22,
          '김순둥': 22
          }
>>> dc
{'이순둥': 22, '정순둥': 22, '김순둥': 22}
```

```
>>> dc = {
          '이순둥': 22,
          '이순둥': 23,
          '이순둥': 24
          }
>>> dc
{'이순둥': 24}
```

4-1. 정의

- 지금까지 배운 데이터 타입
  - int형 데이터 예) 3, 5, 100
  - float형 데이터 예) 2.2 100.0 3.14159
  - 문자열 데이터 예) "I am a boy", "Girl", "Taejun"
  - 리스트 데이터 예) [1, 2, 3], ["AB", "CD], [2.5, "AB"]
  - 부울형 데이터 예) True, False
  - 튜플형 데이터 예) (3, 4, 8, 0), (2.1, 'a', 3)
  - **딕셔너리형 데이터 예) {'name': 'john', 'age': 21}**
  - 레인지(range)

4-2. 딕셔너리 연산

- 딕셔너리에서 값을 가져오고 값을 바꾸기

```
>>> dc = {
          '코카콜라': 900,
          '바나나맛우유': 750,
          '비타500': 600,
          '삼다수': 450
          }
```

```
>>> v = dc['삼다수']
>>> v
450
```

튜플이나 리스트의 인덱싱 연산과 비슷하나,
인덱스 값이 '숫자'가 아닌 '키'를 이용함

```
>>> dc['삼다수'] = 550
>>> dc
{'코카콜라': 900, '바나나맛우유': 750, '비타500': 600, '삼다수': 550}
```

4-2. 딕셔너리 연산

- 딕셔너리에서 데이터 추가 및 삭제

```
>>> dc['삼다수'] = 550
>>> dc
{'코카콜라': 900, '바나나맛우유': 750, '비타500': 600, '삼다수': 550}
```

데이터 추가는 값의 수정과 동일하다.
키가 없는 상황에서 수정과 동일한 연산을 하면 키와 값이 추가된다.

```
>>> dc['카페라떼'] = 1300
>>> dc
{'코카콜라': 900, '바나나맛우유': 750, '비타500': 600, '삼다수': 550,
'카페라떼': 1300}
```

```
>>> del dc['비타500']
>>> dc
{'코카콜라': 900, '바나나맛우유': 750, '삼다수': 550, '카페라떼':
1300}
```

데이터 삭제는 리스트의 값 삭제 방법 del과 동일하다.

4-2. 딕셔너리 연산

- '==' 연산자로 알아보는 딕셔너리의 특징
  → 값의 저장 순서가 중요하지 않음

```
>>> t1 = [1, 2, 3]
>>> t2 = [1, 2, 3]
>>> t3 = [3, 2, 1]
>>> t1 == t2
True
>>> t1 == t3
False
```

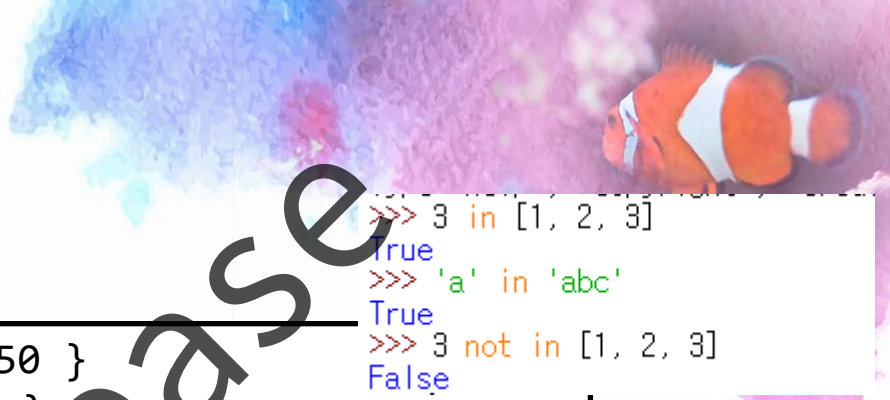
리스트는 저장된 값과 순서가 모두 같아야
같은 리스트로 인정한다.

```
>>> d1 = {1: 'a', 2: 'b'}
>>> d2 = {1: 'a', 2: 'b'}
>>> d3 = {2: 'b', 1: 'a'}
>>> d1 == d2
True
>>> d1 == d3
True
```

딕셔너리의 데이터는 저장 순서가 의미가 없다.
딕셔너리는 저장 순서가 의미 없는 데이터를
담기 위해 만들어진 데이터 타입이다.

저장 순서가 중요하다면 딕셔너리에
데이터를 담으면 안된다.

4-2. 딕셔너리 연산



- in 연산자: 딕셔너리는 '키'를 기준으로 함

```
>>> dc1 = {‘코카콜라’: 900, ‘삼다수’: 450 }
>>> dc2 = {‘새우깡’: 700, ‘콘치즈’: 850 }
```

새우깡 가격이 올랐다!

```
>>> dc2[‘새우깡’] = 950    # dc2에 과자정보가 담겨 있으므로 OK
```

문제: 실수로 음료 정보가 담긴 dc1을 대상으로 한다면? 새로운 값이 추가되는 꼴이다.

```
>>> dc1[‘새우깡’] = 950    # 실수로 dc1에 접근
```

안전하게 코딩하라! → in 연산을 통해 딕셔너리에 특정 키가 있는지 확인하고 수정하라.

```
>>> if '새우깡' in dc2:
        dc2['새우깡'] = 950      # 수정
```

안전하게 코딩하라! → not in 연산을 통해 딕셔너리에 특정 키가 없는지 확인하고 추가하라.

```
>>> if '카페라떼' not in dc1:
        dc1['카페라떼'] = 1200 # 추가
```

4-3. 딕셔너리와 for문

```
>>> dc = {'새우깡': 700, '콘치즈': 850, '꼬깔콘': 750}
>>> for i in dc:
        print(i, end = ' ')

새우깡  콘치즈  꼬깔콘
```

i에 저장되는 것은 '키'이다!

딕셔너리의 '키'를 대상으로 for 루프가 돌아간다.

과자의 가격을 모두 70원 인상시킨다면, 아래와 같이 코드를 구성해야 한다.

```
>>> dc = {'새우깡': 700, '콘치즈': 850, '꼬깔콘': 750}
>>> for i in dc:
        dc[i] += 70

>>> dc
{'새우깡': 770, '콘치즈': 920, '꼬깔콘': 820}
```

연습 문제

- 예제 11. 과자의 정보가 담겨 있는
  아래 딕셔너리에 "'홈런볼' : 900"을 추가하라.

```
>>> dc = {'새우깡': 700, '콘치즈': 850, '꼬깔콘': 750}
```

- 예제 12. '예제 11'에 이어 모든 과자의 가격을
  100원 인상하라.

- 예제 13. '예제 12'에 이어 '콘치즈' 과자 이름이
  '치츠콘'으로 변경되었다.
  따라서, 콘치즈를 삭제하고
  새로운 정보를 추가하라.
  '치즈콘' : 950

실전 문제

- 문제 1. 서로 다른 n개에서 r개를 택해 일렬로 나열하는 수를 순열(permutation)이라 하고, nPr이라고 한다. (저번 실전 문제 응용)
  이 를 permutation 함수로 만들어라.

$$n(n-1)(n-2)\ldots(n-r+1)$$

  n과 r은 사용자가 입력하게 하라.
  예시) n = 10, r = 3일 때, 순열 값은 720이다.

- 문제 2. 위 함수를 my_math라는 모듈로 만들고 이를 불러와서 실행하라.

- 문제 3. "예제 5 ~ 7"까지를 my_math_2라는 모듈로 만들고 이를 불러와서 실행하라.

05

주사위 게임 만들기

1. 게임 보드 표시
2. 말 옮기기
3. 주사위 던지기
4. 승패 확인

5-1. 게임 보드 표시

- 왼쪽부터 시작해서 전체 30개짜리 별(*) 표시
- P는 플레이어, C는 컴퓨터
- S는 끝까지 가서 이겼다는 의미

```
**************P**************S
*******C**********************S
```

```
player_pos = 6
def board():
        print('*' * (player_pos - 1) + "P" + "*" * (30 - player_pos))

board()
```

```
*****P************************
```

5-1. 게임 보드 표시

- 왼쪽부터 시작해서 전체 30개짜리 별(*) 표시
- P는 플레이어, C는 컴퓨터
- S는 끝까지 가서 이겼다는 의미

```
player_pos = 6
computer_pos = 3
def board():
        print('*' * (player_pos - 1) + "P" + "*" * (30 - player_pos))
        print('*' * (computer_pos - 1) + "C" + "*" * (30 - computer_pos))

board()
```

```
*****P************************
**C*************************
```

5-1. 게임 보드 표시

- 개발자(사람)은 똑같은 일, 또 하는 것 싫어 합니다! (귀찮음~)

```
def board(name, pos):
        print('*' * (pos - 1) + name + "*" * (30 - pos))

board("P", 6)
board("C", 3)
```

```
*****P************************
**C*************************
```

5-2. 말 옮기기

- 말을 얼마나 움직일까?

```
player_pos = 1
computer_pos = 1

def board(name, pos):
        print('*' * (pos - 1) + name + "*" * (30 - pos))

while True:
        board("P", player_pos)
        board("C", computer_pos)
        input("Enter 키를 입력하면 말이 움직여요~")
        player_pos += 1
        computer_pos += 2
```

5-3. 주사위 던지기

- 주사위는 1~6까지 뭐가 나올지 모름! → 난수(random number)

```
from random import randint
# import random으로 쓰면 random.randint(1, 6)으로 써야 함

player_pos = 1
computer_pos = 1

def board(name, pos):
        print('*' * (pos - 1) + name + "*" * (30 - pos))

while True:
        board("P", player_pos)
        board("C", computer_pos)
        input("Enter 키를 입력하면 말이 움직여요~")
        player_pos += randint(1, 6)
        computer_pos += randint(1, 6)
```

5-3. 주사위 던지기

- 주사위(난수) 결과가 얼마인지 확인

```
while True:
        board("P", player_pos)
        board("C", computer_pos)
        input("Enter 키를 입력하면 말이 움직여요~")
        player_rand = randint(1, 6)
        computer_rand = randint(1, 6)
        print("Player:", player_rand, "Computer:", computer_rand)
        player_pos += player_rand
        computer_pos += computer_rand
```

5-4. 승패 확인

- 승리했는지는 if(조건문)로 확인 (최종 코드)

```
from random import randint
# import random으로 쓰면 random.randint(1, 6)으로 써야 함

player_pos = 1
computer_pos = 1

def board(name, pos):
        print('*' * (pos - 1) + name + "*" * (30 - pos) + "V")
```

5-4. 승패 확인

- 승리했는지는 if(조건문)로 확인 (최종 코드)

```
print("Game Start!")
board("P", player_pos)
board("C", computer_pos)

while True:
    input("Enter 키를 입력하면 플레이어의 말이 움직여요~")
    player_rand = randint(1, 6)
    print("Player:", player_rand)
    player_pos += player_rand

    if player_pos >= 30:
        player_pos = 30

    board("P", player_pos)
    if player_pos >= 30:  # 난수가 30이 넘어가면 이긴 것으로 판정
        print("Player 승리!")
        break
```

5-4. 승패 확인

- 승리했는지는 if(조건문)로 확인 (최종 코드)

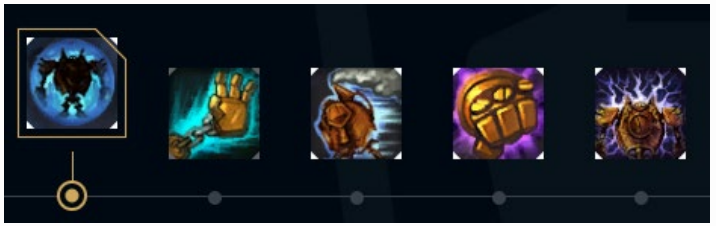
```
# while 문 안임 (이전 페이지 이어서)
    input("Enter 키를 입력하면 컴퓨터의 말이 움직여요~")
    computer_rand = randint(1, 6)
    print("Computer:", computer_rand)
    computer_pos += computer_rand

    if computer_pos >= 30:
        computer_pos = 30

    board("C", computer_pos)
    if computer_pos == 30:   # 난수가 30이 넘어가면 이긴 것으로 판정
        print("Computer 승리!")
        break
```

실전 문제

- 문제 4. 앞서 만든 주사위 게임에는 같은 로직이 반복한다.
  이를 더 심플하게 하기 위해 함수로 만들어서 코드를 더욱 단순히 하라.

- 문제 5. 위에서 만든 함수를 모듈(이름은 예쁘게 정해서)에 넣고
  이를 불러와서 핵심 코드(while 문 안)를 단순히 하라.

- 실제 게임에서는 어떻게 구현하는가? 클래스(변수 + 함수) → 심화



다음 시간

1. **사라진 문자 찾기 게임**
   ---------- GUI ----------
2. tkinter 기초 - 라벨과 버튼
3. tkinter 기초 - 캔버스
4. tkinter 기초 - 뽑기 프로그램
5. tkinter 기초 - 텍스트 입력 필드
6. tkinter 기초 - 체크 버튼과 메시지 박스

참고 자료

- 윤성우, 열혈 파이썬(기본편/중급편), 오렌지미디어, 2017.
- 히로세 츠요시, 파이썬으로 배우는 게임 개발 (입문편/실전편), 제이펍, 2020.
- 폴 데이텔, 하비 데이텔, 안진섭, 프로그래머를 위한 Python, 성안당, 2021.
- 루시아누 하말류, 전문가를 위한 파이썬, O'REILLY, 2016.
- 브렛 슬라킨, Effective Python, 2nd edition, 길벗, 2020.