

**산업 인공지능 - 실습 2**

# **Python 프로그래밍**

2021 Spring

# 1. 반복문

## ❖ 반복문 (repetition statement)

- 반복적인 작업을 실행

```
print("환영합니다.")  
print("환영합니다.")  
print("환영합니다.")  
print("환영합니다.")  
print("환영합니다.")
```

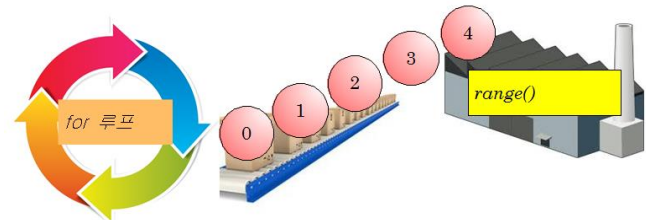
```
for x in range(5) :  
    print("환영합니다.")
```

`range(n)` : 0부터 n-1까지 정수 생성

`range(5)` ⇒ 0, 1, 2, 3, 4 순차적 생성

### ■ 반복문의 종류

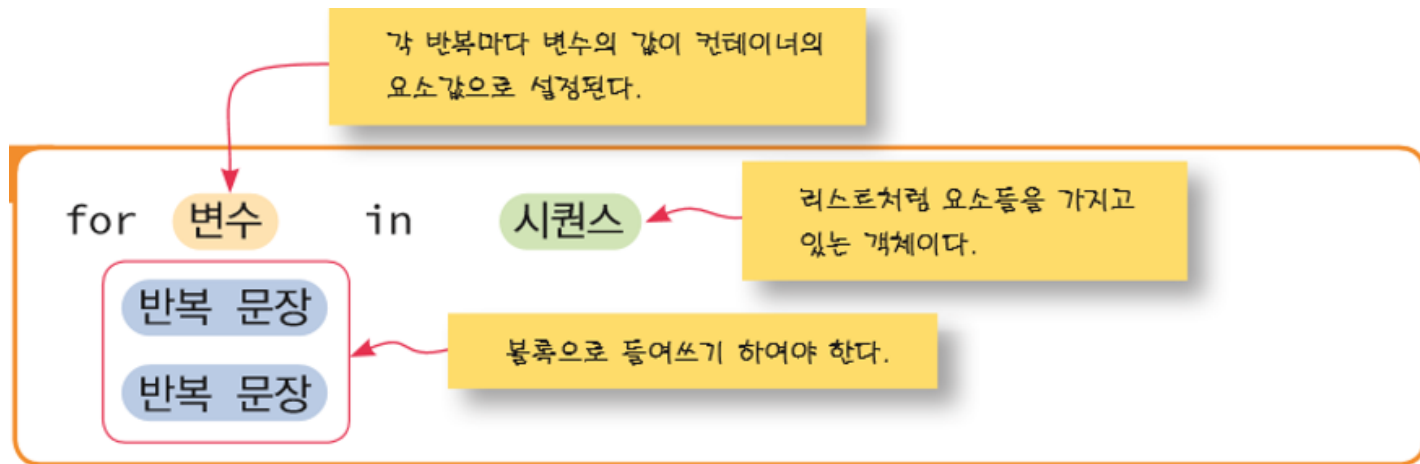
- `for` 문 - 정해진 횟수만큼 반복
- `while` 문 - 어떤 조건이 만족되는 동안, 반복을 계속



## 2. for 문

### ❖ for 문

- 정해진 횟수만큼 반복



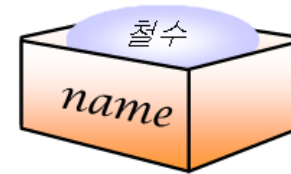
```
for name in ["철수", "영희", "길동", "유신"]:  
    print("안녕! " + name)
```

```
안녕! 철수  
안녕! 영희  
안녕! 길동  
안녕! 유신
```

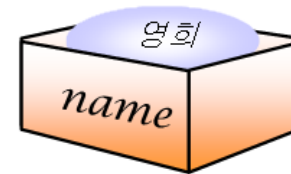
# for 문

```
for name in ["철수", "영희", "길동", "유신"]:  
    print("안녕! " + name)
```

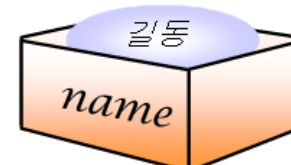
```
for name in ["철수", "영희", "길동", "유신"]:  
    print("안녕! " + name)
```



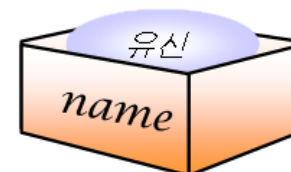
```
for name in ["철수", "영희", "길동", "유신"]:  
    print("안녕! " + name)
```



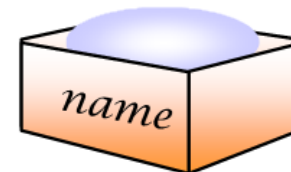
```
for name in ["철수", "영희", "길동", "유신"]:  
    print("안녕! " + name)
```



```
for name in ["철수", "영희", "길동", "유신"]:  
    print("안녕! " + name)
```



```
for name in ["철수", "영희", "길동", "유신"]:  
    print("안녕! " + name)
```



# for 문

```
for x in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]:  
    print(x, end=" ")
```

```
0 1 2 3 4 5 6 7 8 9
```

# for 문

## ❖ range( ) 함수

```
range( [ start ,] stop [, step ])
```

- start부터 stop-1까지 step의 간격으로 정수들을 생성
- 대괄호 부분 생략 가능
- start이 생략되면 0으로 간주
- step이 생략되면 1로 간주

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

45

```
sum = 0
for x in range(0, 10) :
    sum = sum + x
print(sum)
```

45

# 예제: 정수들의 합

- ❖ 1부터 사용자가 입력한 수 n까지 더해서 출력하는 프로그램 (for 문 사용)

어디까지 계산할까요: 10  
1부터 10 까지의 정수의 합= 55

```
# 반복을 이용한 정수합 프로그램
sum = 0

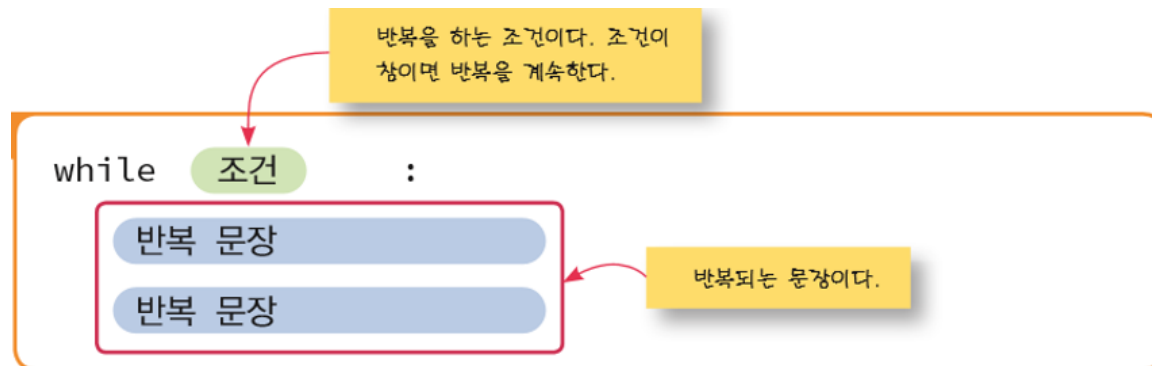
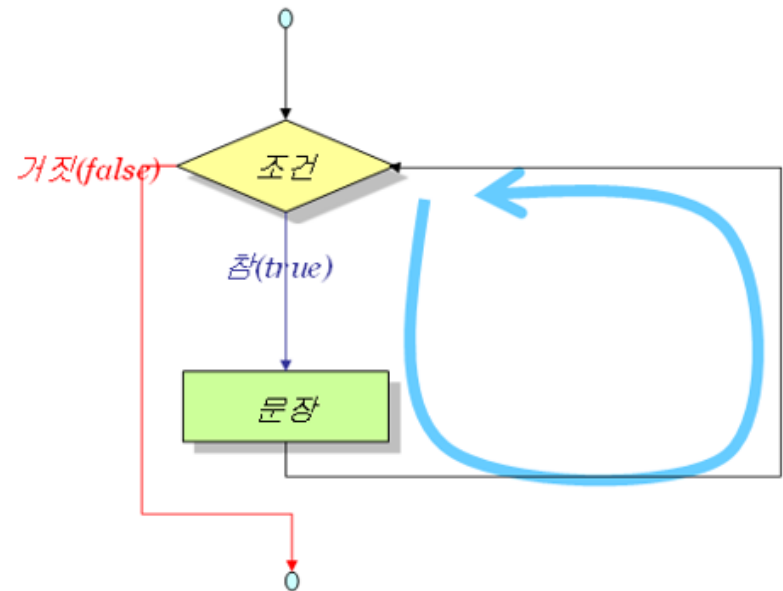
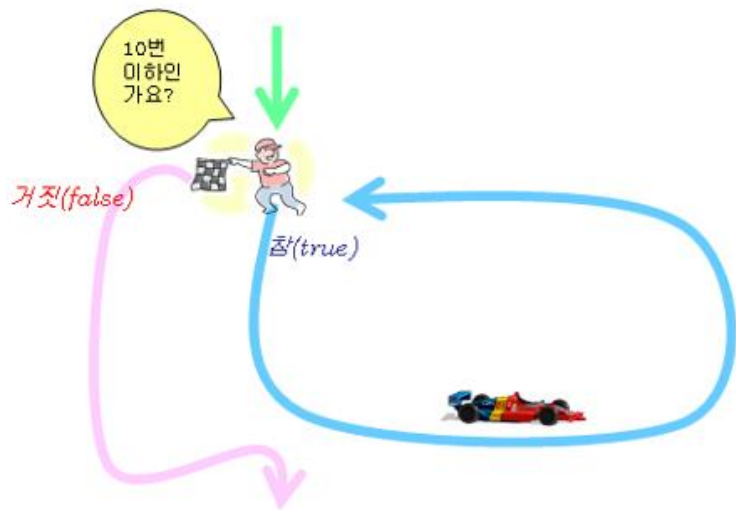
limit = int(input("어디까지 계산할까요: "))
for i in range(1, limit+1):
    sum += i

print("1부터 ", limit, "까지의 정수의 합= ", sum)
```

# 3. while 문

## ❖ while 문

- 조건을 정해놓고 반복





# while 문

```
i = 0;  
while i < 5 :  
    print("환영합니다.")  
    i = i + 1  
print("반복이 종료되었습니다.")
```

```
환영합니다.  
환영합니다.  
환영합니다.  
환영합니다.  
환영합니다.  
반복이 종료되었습니다.
```

# 예제: 함수 그래프 그리기

- ❖ 0, 1, 2, ..., 9까지를 차례대로 화면에 출력하는 프로그램을 작성하여 보자

```
0 1 2 3 4 5 6 7 8 9
```

```
i = 0
while i < 10:
    print (i, end=" ");
    i = i + 1
```

# 예제: $(1+2+3+\dots+9+10)$ 계산하기

❖  $(1+2+3+\dots+9+10)$ 의 값을 계산하는 프로그램을 작성하여 보자.

합계 = 55

```
i = 1
sum = 0;

while i <= 10:
    sum = sum + i
    i = i + 1
print("합계=", sum)
```

# 예제: 팩토리얼 계산

$$\diamond n! = 1 \times 2 \times 3 \times \dots \times (n-1) \times n$$

10!은 3628800입니다.

```
i = 1
factorial = 1

while (i <= 10):
    factorial = factorial * i
    i = i + 1
print ("10!은 %d입니다." % factorial)
```

# 예제: 구구단 3단 출력

❖ 구구단 중에서 3단을 반복문을 이용하여 출력하여 보자

```
3*1 = 3  
3*2 = 6  
3*3 = 9  
3*4 = 12  
3*5 = 15  
3*6 = 18  
3*7 = 21  
3*8 = 24  
3*9 = 27
```

```
i = 1  
while i <= 9:  
    print("3*%d = %d" % (i, 3*i))  
    i = i + 1
```

# 예제: 배수의 합 계산 프로그램

❖ 1부터 100사이의 모든 3의 배수의 합을 계산하여 출력하는 프로그램

1부터 100 사이의 모든 3의 배수의 합은 1683입니다.

```
sum = 0
number = 1

while number <= 100 :
    if number %3 == 0 :
        sum = sum + number
        number = number + 1
print("1부터 100 사이의 모든 3의 배수의 합은 %d입니다." % sum)
```

# 예제: 자리수의 합

- ❖ 정수 안의 각 자리수의 합을 계산하는 프로그램을 작성해보자.  
예를 들어서 1234라면  $(1+2+3+4)$ 를 계산하는 것이다.

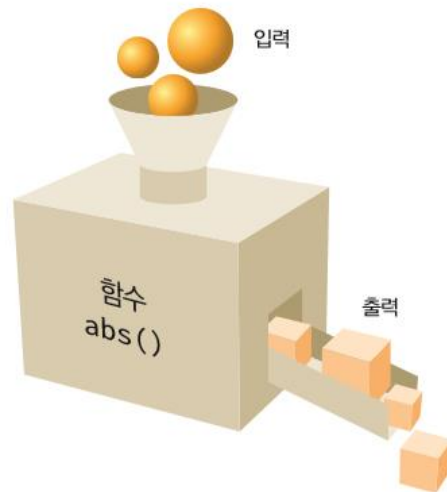
자리수의 합은 10입니다.

```
number = 1234
sum = 0;
while number > 0 :
    digit = number % 10
    sum = sum + digit
    number = number // 10
print("자리수의 합은 %d입니다." % sum)
```

# 1. 함수

## ❖ 함수 (function, 函數)

- 특정 작업을 수행하는 명령어들의 모음에 이름을 붙인 것
- 작업에 필요한 데이터를 전달받을 수 있음
- 작업이 완료된 후에는 작업의 결과를 호출자에게 반환 가능



```
print( )  
input( )  
abs( )  
...
```

- 함수 호출(call)
  - 함수 안의 명령어들 실행

```
>>> value = abs(-100)  
>>> value  
100
```



# 함수

## ❖ 함수의 필요성

### ▪ 반복되는 코드의 간단한 표현

- 코드 길이 축소, 가독성(可讀性) 개선, 코드 수정 편리, 복잡한 작업 분해하여 표현

비슷한 코드인데 하나로 합칠 수 있을까?



```
sum = 0;
for i in range(1, 11)
    sum += i;
```

```
sum = 0;
for i in range(1, 21)
    sum += i;
```

```
get_sum(1, 10)
```

```
get_sum(1, 20)
```

```
def get_sum(start, end)
    sum = 0;
    for i in range(start, end+1)
        sum += i;
    return sum
```

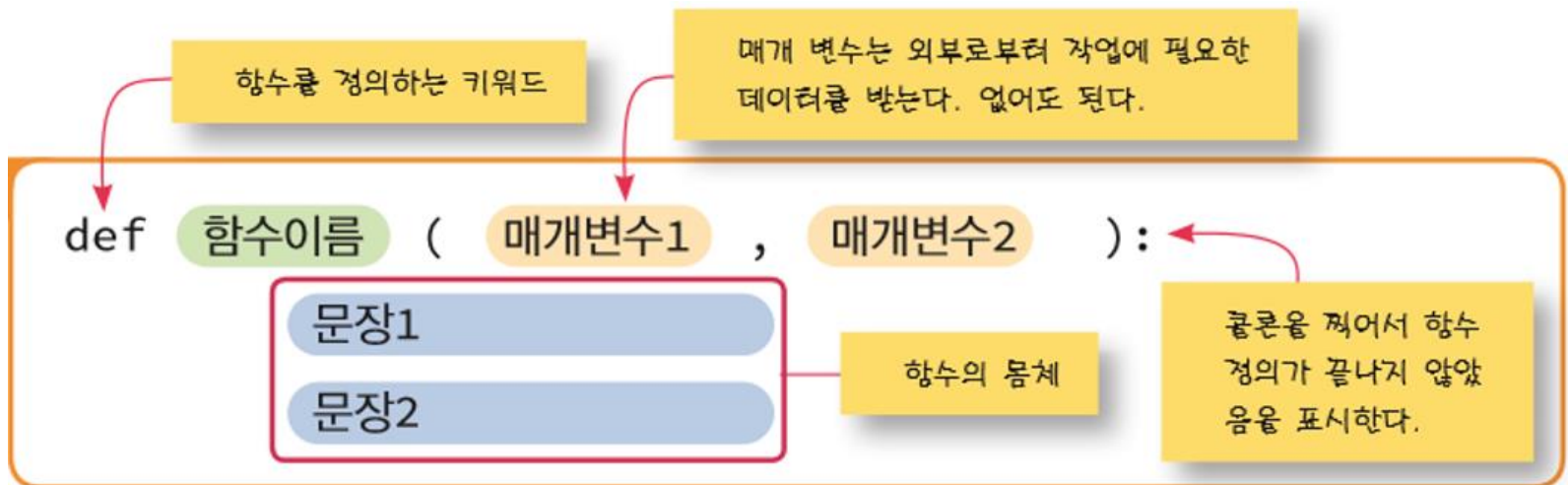
함수를 사용하면 됩니다.



## 2. 함수 정의

### ❖ 함수 정의

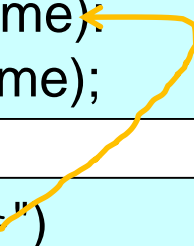
- 헤더(header)와 몸체(body)로 구성
- 헤더 : def로 시작 (define, 정의하다)
  - **def** 함수이름(매개변수) :
  - 함수이름 : 함수를 나타내는 식별자
  - 매개변수(parameter) : 외부에서 함수에 전달하는 값 전달
- 몸체 : 함수가 수행하는 작업을 위한 문장들의 집합



# 함수 정의


## ❖ 함수 정의와 호출의 예

```
>>>def say_hello(name):  
    print("안녕, ", name);
```



```
>>>say_hello( " 민수")  
안녕, 민수
```

```
>>>def say_hello(name, msg):  
    print("안녕, ", name, "야, ", msg);
```



```
>>>Name = " 민수"  
>>>Msg = "어서 집에 오너라"  
>>>say_hello(Name, Msg)  
안녕, 민수야, 어서 집에 오너라
```

# 함수 정의

## ❖ 값을 반환하는 함수

- `return` 문 사용
- 호출한 곳에 값 전달

```
>>>def get_sum(start, end) :  
    sum = 0  
    for i in range(start, end+1) :  
        sum += i  
    return sum
```

```
>>>value = get_sum(1, 10)  
>>>print(value)  
55
```

# 함수 정의

## ❖ 함수 이름

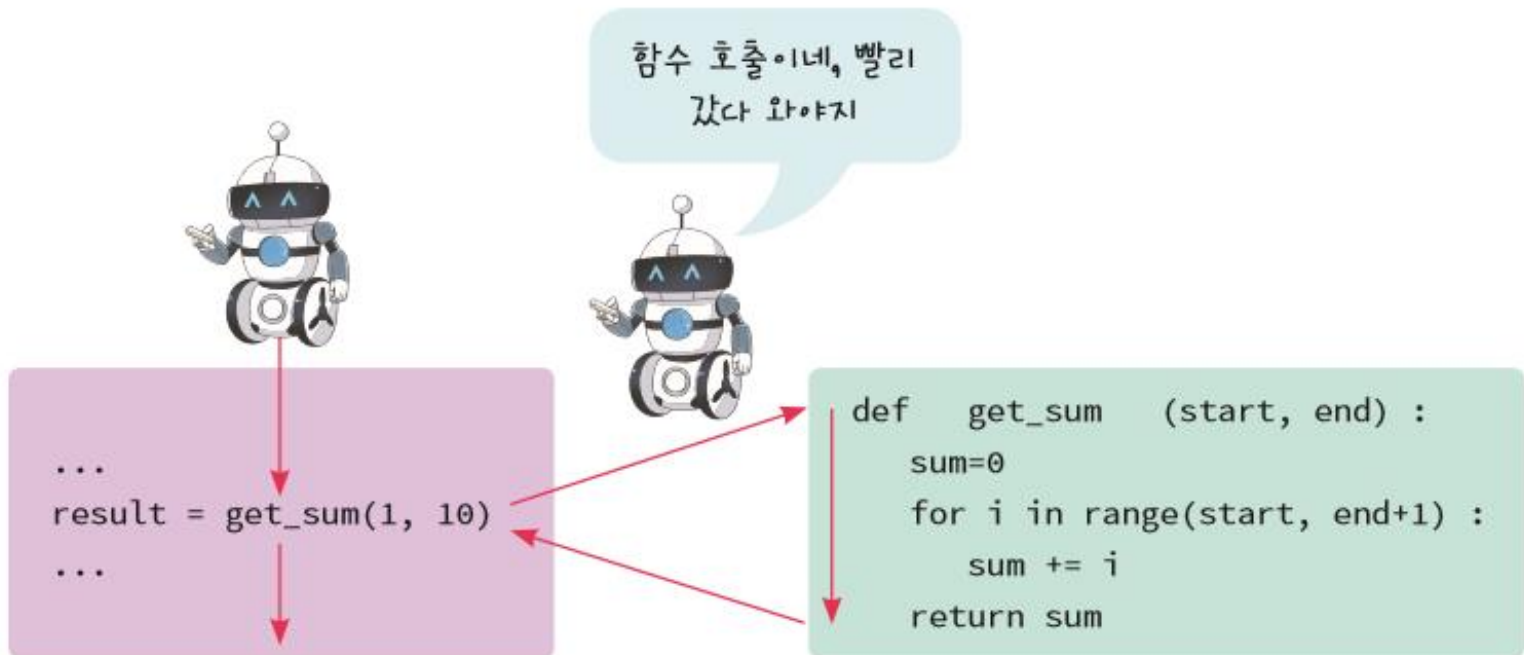
- 식별자에 대한 규칙 준수
- 함수의 목적을 설명하는 동사 또는 동사+명사를 사용

square(side)	// 정수를 제공하는 함수
compute_average(list)	// 평균을 구하는 함수
set_cursor_type(c)	// 커서의 타입을 설정하는 함수

# 함수 정의

## ❖ 함수 호출(function call)

- 함수이름과 전달할 값에 대한 정보를 써주면 됨
- 함수 안의 문장들은 호출되지 전에는 미실행
- 호출직후 문장들 순차적으로 실행
- 실행 완료 후에는 호출한 위치로 복귀



# 함수 정의

## ❖ 함수 호출

- 동일한 함수의 여러 번 호출 가능



```
...  
x = get_sum(1, 10)  
...  
y = get_sum(1, 20)
```

```
def get_sum (start, end) :  
    sum=0  
    for i in range(start, end+1) :  
        sum += i  
    return sum
```

```
def get_sum(start, end) :  
    sum=0  
    for i in range(start, end+1) :  
        sum += i  
    return sum
```

```
print( get_sum(1, 10))  
print( get_sum(1, 20))
```

```
55  
210
```

# 예: 온도 변환

❖ 섭씨 온도를 화씨 온도로 변환하여 반환하는 함수 FtoC()

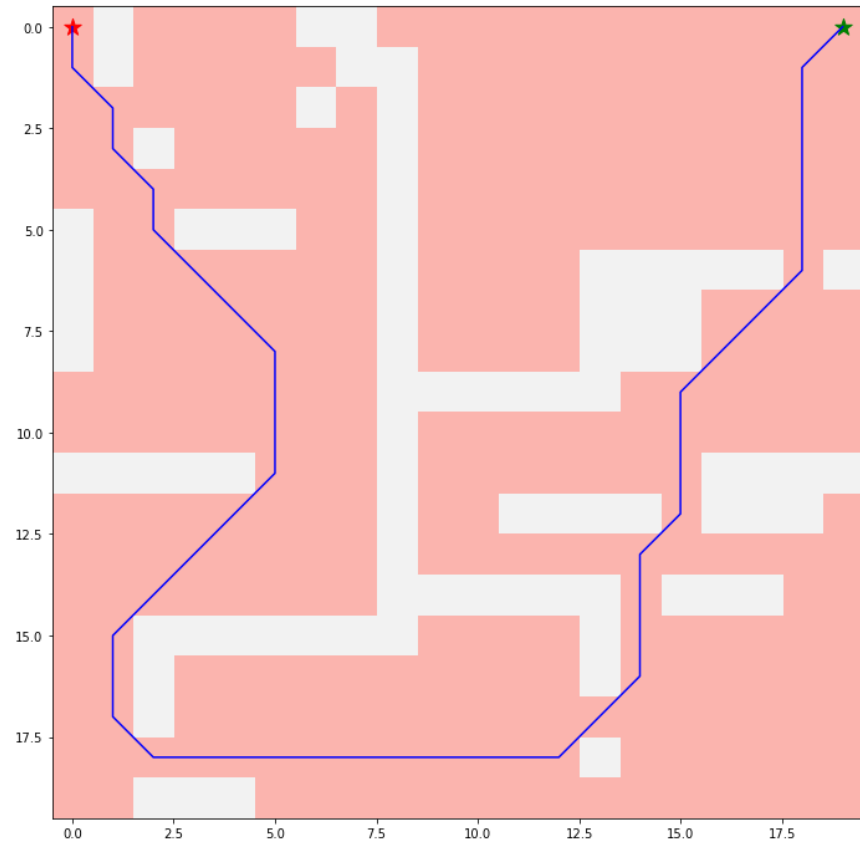
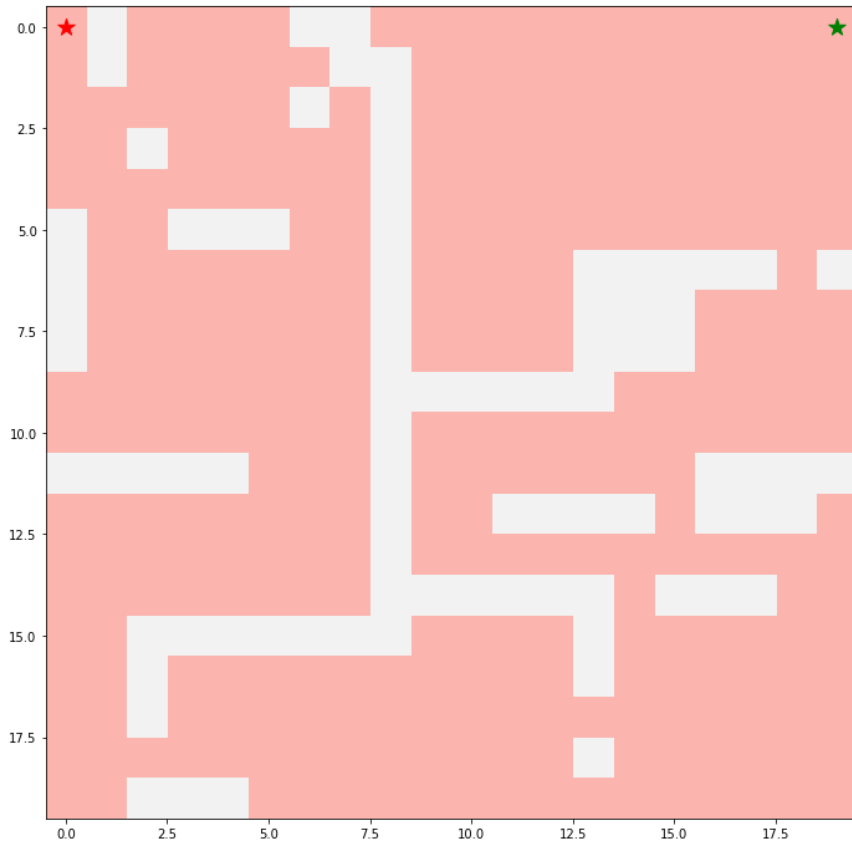
화씨온도를 입력하시오: 32.0  
0.0

```
# 함수가 여기서 정의된다.  
def FtoC(temp_f):  
    temp_c = (5.0 * (temp_f - 32.0)) / 9.0;  
    return temp_c;  
  
temp_f = float(input("화씨온도를 입력하시오: "))  
  
# FtoC() 함수를 호출한다.  
print(FtoC(temp_f))
```



# 프로그래밍 실습 : A\* 알고리즘

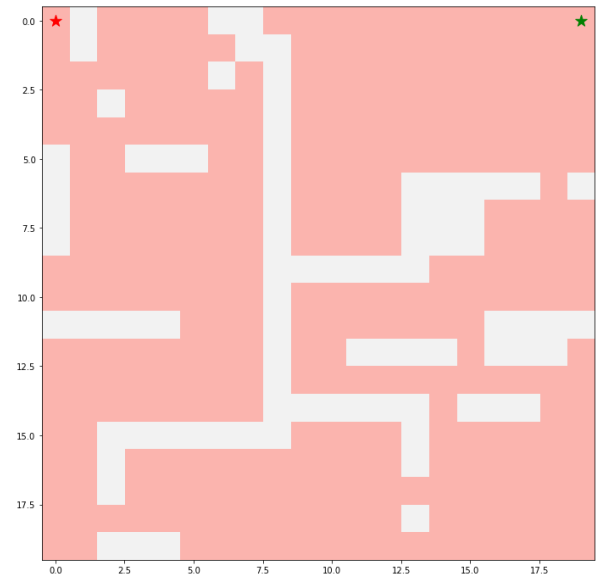
## ❖ 격자 공간에서 최단 경로 찾기



```

1 import numpy as np
2 import heapq # min heap을 구현하는 heap queue
3 import matplotlib.pyplot as plt
4 from matplotlib.pyplot import figure
5
6 # 지도 1:벽, 0: 빈공간
7 grid = np.array([
8     [0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
9     [0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
10    [0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
11    [0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
12    [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
13    [1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
14    [1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1],
15    [1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0],
16    [1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0],
17    [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
18    [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
19    [1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1],
20    [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1],
21    [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
22    [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0],
23    [0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
24    [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
25    [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
26    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
27    [0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])
28
29 start = (0,0) # 시작 위치
30 goal = (0,19) # 목적지 위치
31
32 # 휴리스틱 함수 h() : a와 b사이의 유클리드 거리
33 def heuristic(a, b):
34     return np.sqrt((b[0] - a[0]) ** 2 + (b[1] - a[1]) ** 2)
35

```



36 # A\* 알고리즘

37 def Astar(array, start, goal):

38 neighbors = [(0,1),(0,-1),(1,0),(-1,0),(1,1),(1,-1),(-1,1),(-1,-1)] # 이웃 위치

39 close\_set = set() # 탐색이 종료된 위치들의 집합

40 came\_from = {}

41 gscore = {start:0} # 시작 위치의 g() 값

42 fscore = {start:heuristic(start, goal)} # 시작위치의 f() 값

43 oheap = [] # min-heap

44 heapq.heappush(oheap, (fscore[start], start)) # (거리, 출발지) min-heap에 저장

45

46 while oheap:

47 current = heapq.heappop(oheap)[1] # f()값이 최소인 노드 추출

48 if current == goal: # 목적지 도착

49 data = []

50 while current in came\_from: # 목적지에서 역순으로 경로 출력

51 data.append(current)

52 current = came\_from[current]

53 return data

54 close\_set.add(current) # current 위치를 탐색이 종료된 것으로 간주

55

56 for i, j in neighbors: # current 위치의 각 이웃 위치에 대해 f() 값 계산

57 neighbor = current[0] + i, current[1] + j # 이웃 위치

58 if 0 <= neighbor[0] < array.shape[0]:

59 if 0 <= neighbor[1] < array.shape[1]:

60 if array[neighbor[0]][neighbor[1]] == 1: # 벽

61 continue

62 else: # y 방향의 경계를 벗어난 상황

63 continue

64 else: # x 방향의 경계를 벗어난 상황

65 continue

66

67 temp\_g\_score = gscore[current] + heuristic(current, neighbor) #  $g^*(n) = g(c) + h(c, n)$

68 if neighbor in close\_set and temp\_g\_score >= gscore.get(neighbor, 0):

69 continue # 이미 방문한 위치이면서  $g^*(n)$  값이 기존  $g(n)$  값보다 큰 경우 --> 무시

70

71 if temp\_g\_score < gscore.get(neighbor, 0) or neighbor not in [i[1] for i in oheap]:

72 #  $g^*(n) < g(n)$  이거나, n을 처음 방문한 경우

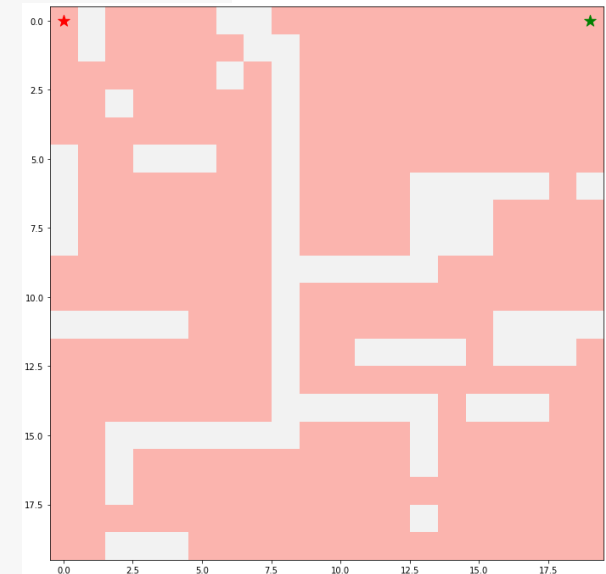
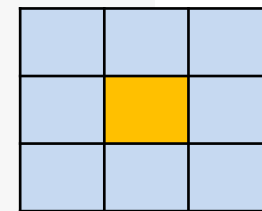
73 came\_from[neighbor] = current # neighbor에 도달한 최선의 경로에서 직전 위치는 current

74 gscore[neighbor] = temp\_g\_score #  $g(n) = g^*(n)$

75 fscore[neighbor] = temp\_g\_score + heuristic(neighbor, goal) #  $f(n) = g(n) + h(n)$

76 heapq.heappush(oheap, (fscore[neighbor], neighbor)) # min heap에 (f(), neighbor) 삽입

77 return False



$$\hat{f}(n) = g(n) + \hat{h}(n)$$

```

78
79 route = Astar(grid, start, goal)
80 route = route + [start] # 출발 위치 추가
81 route = route[::-1] # 역순으로 변환
82 print('경로: ', route)
83
84 # route에서 x와 y 좌표 추출
85 x_coords = []
86 y_coords = []
87 for i in (range(0,len(route))):
88     x = route[i][0]
89     y = route[i][1]
90     x_coords.append(x)
91     y_coords.append(y)
92
93 # 지도와 경로 그리기
94 fig, ax = plt.subplots(figsize=(12,12))
95 ax.imshow(grid, cmap=plt.cm.Pastel1)
96 ax.scatter(start[1],start[0], marker = "*", color = "red", s = 200)
97 ax.scatter(goal[1],goal[0], marker = "*", color = "green", s = 200)
98 ax.plot(y_coords,x_coords, color = "blue")
99 plt.show()

```

경로: [(0, 0), (1, 0), (2, 1), (3, 1), (4, 2), (5, 2), (6, 3), (7, 4), (8, 5), (9, 5), (10, 5), (11, 5), (12, 4), (13, 3), (14, 2), (15, 1), (16, 1), (17, 1), (18, 2), (18, 3), (18, 4), (18, 5), (18, 6), (18, 7), (18, 8), (18, 9), (18, 10), (18, 11), (18, 12), (17, 13), (16, 14), (15, 14), (14, 14), (13, 14), (12, 15), (11, 15), (10, 15), (9, 15), (8, 16), (7, 17), (6, 18), (5, 18), (4, 18), (3, 18), (2, 18), (1, 18), (0, 19)]

