데이터분석 프로그래밍
반복문

임현기

반복문의 필요성

- 회사에 중요한 손님이 오셔서 대형 전광판에 환영인사를 5번 출력해야하는 상황

```
파이썬 주식회사의 방문을 환영합니다!
파이썬 주식회사의 방문을 환영합니다!
파이썬 주식회사의 방문을 환영합니다!
파이썬 주식회사의 방문을 환영합니다!
파이썬 주식회사의 방문을 환영합니다!
```

- 반복 구조를 사용하지 않는다면 다음과 같이 동일한 문장을 복사해야 함

```
print("파이썬 주식회사의 방문을 환영합니다!")
print("파이썬 주식회사의 방문을 환영합니다!")
print("파이썬 주식회사의 방문을 환영합니다!")
print("파이썬 주식회사의 방문을 환영합니다!")
print("파이썬 주식회사의 방문을 환영합니다!")
```

반복문의 필요성

- 100번 반복한다면?

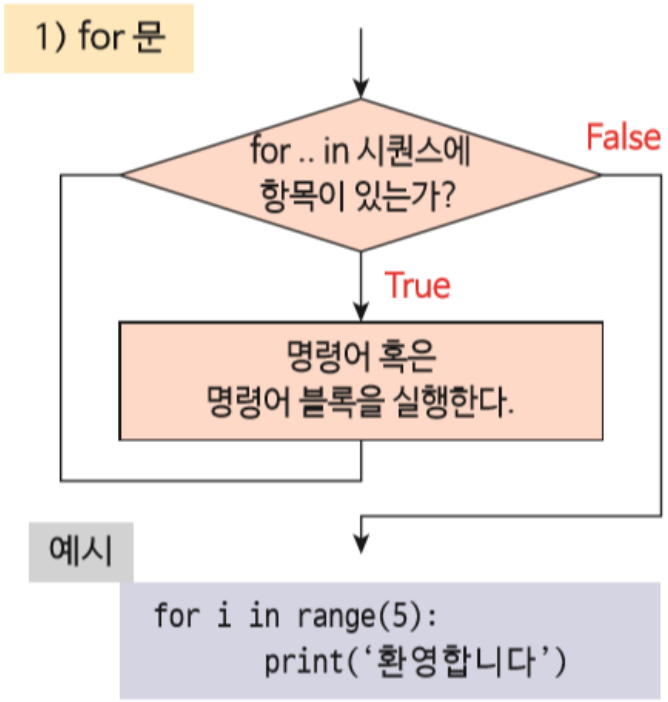
```
for i in range(100):
    print("파이썬 주식회사의 방문을 환영합니다!")
```

반복문의 종류

- for 문: 보통 횟수를 정해놓고 반복
  - 항목들을 모아놓은 시퀀스라는 객체가 있음
  - 항목을 하나씩 가져와서 반복할 때 적합

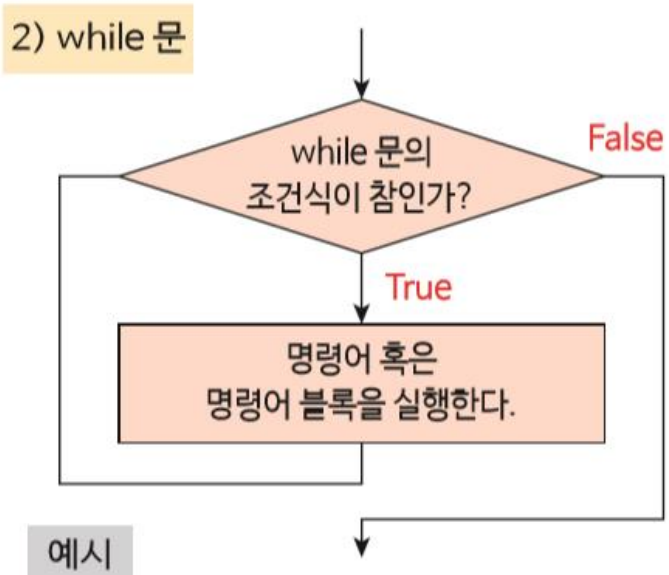
- while 문: 특정 조건이 만족되면 계속 반복

반복문의 종류



- range(5)라는 함수는 0, 1, 2, 3, 4의 숫자 시퀀스를 자동으로 생성하는 함수
- 0에서 4까지의 숫자를 반환하는 작업을 끝낼 때까지 반복

반복문의 종류



예시

```
i = 0
while i < 5:
    print('환영합니다')
    i = i + 1
```

- i를 0으로 초기화 시킴
- while문은 i < 5의 조건이 참일 경우 환영합니다를 출력하고 i 값을 1 증가
- 최초의 i 값이 0이므로 5회 반복 수행

for 루프

```
for i in [1, 2, 3, 4, 5]:          #  반복문의 끝에 :이 있어야 함
    print("방문을 환영합니다!")     # if 문과 마찬가지로 들여쓰기를 하여야 함
```

– [...]은 리스트 자료형

for 루프

```
for i in [1, 2, 3, 4, 5]:     # in 뒤에 리스트를 넣고 끝에 :을 넣자
    print("i =", i)           # i 값을 출력해보자
```

– 변수 i값 이용

```
i = 1
i = 2
i = 3
i = 4
i = 5
```

for 루프

- 문자열에 적용

```
for i in "Hello":       # 끝에 :이 있어야 함
    print("i =", i)     # i 값을 출력해보자
```

```
i = H
i = e
i = l
i = l
i = o
```

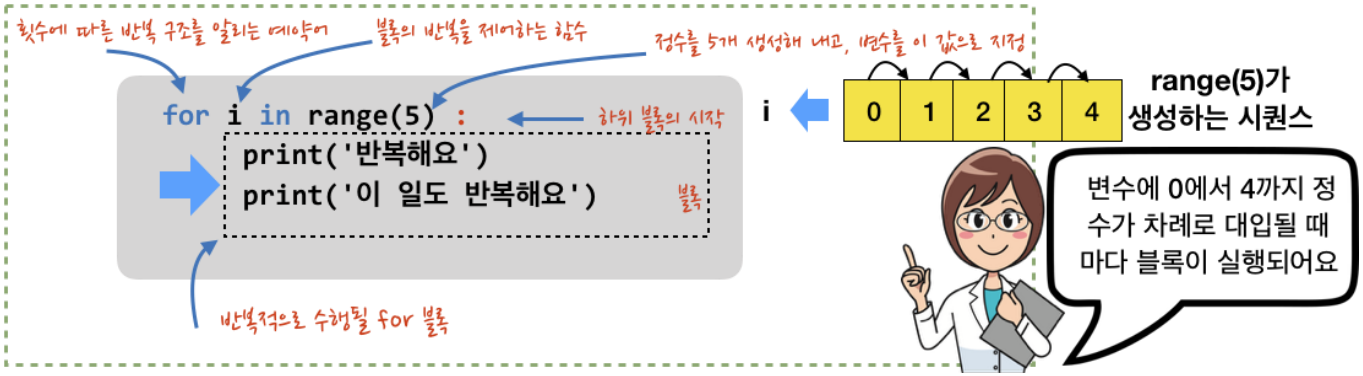
for 루프

```
for i in [1, 2, 3, 4, 5]:          # 1에서 5까지의 리스트
    print("9 *", i, "=", 9 * i )  # 9*i 값을 출력해보자
```

```
9 * 1 = 9
9 * 2 = 18
9 * 3 = 27
9 * 4 = 36
9 * 5 = 45
```

range 함수

- range() 함수




```
for i in range(3):              # (1)
    print('파이썬 주식회사의')    # (2)
    print('방문을 환영합니다!')   # (2)
```

```
파이썬 주식회사의
방문을 환영합니다!
파이썬 주식회사의
방문을 환영합니다!
파이썬 주식회사의
방문을 환영합니다!
```

range 함수

- range(3) 함수는 0, 1, 2까지의 숫자열sequence을 반환
  - 반복할 때마다 변수 i에 이 값들을 대입하면서 문장 반복
  - 첫번째 i는 0, 두번째 i는 1, 마지막 i는 2

- range() 함수에 list() 함수를 적용시키면 range() 함수가 생성하는 정수들을 볼 수 있음

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

range 함수

잠깐 – range() 함수가 반환하는 값은 range 형이다

파이썬의 range() 함수는 range 형의 자료형을 반환한다. range 형의 자료형은 호출이 발생할 때마다 매번 연속된 값을 생성하여 반환하는 특별한 일을 한다. 따라서 주로 for 문과 함께 사용된다.

range 함수

- range 함수는 여러 개의 인자를 이용하여 다양한 정수 시퀀스를 생성할 수 있음

- range(start, stop, step)
  - start에서 시작하여 (stop-1)까지 step 간격으로 정수들을 생성
  - 인자가 1개인 range 함수에서는 stop은 반드시 지정해야 함

range 함수

- range(0, 5, 1): 0, 1, 2, 3, 4까지의 정수가 반환
- range(5): range(0, 5, 1)와 같음

- 0이 아닌 1부터 시작하여서 5까지 반복
  - range(1, 6, 1)

```
for i in range(1, 6, 1):
    print(i, end = " ")     # end = " "로 지정하면 줄바꿈을 하지 않고 공백으로 나열한다.
```

```
1 2 3 4 5
```

- 10부터 시작하여서 1까지 1씩 감소하며 반복
  - range(10, 0, -1)

```
for i in range(10, 0, -1):
    print(i, end = " ")
```

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

사용자로부터 임의의 정수 n을 입력받은 뒤에 for 문을 이용하여서 팩토리얼을 계산해보자. 팩토리얼 n!은 1부터 n까지의 정수를 모두 곱한 것을 의미한다. 즉, n! = 1×2×3×.....×(n-1)×n이다.

원하는 결과

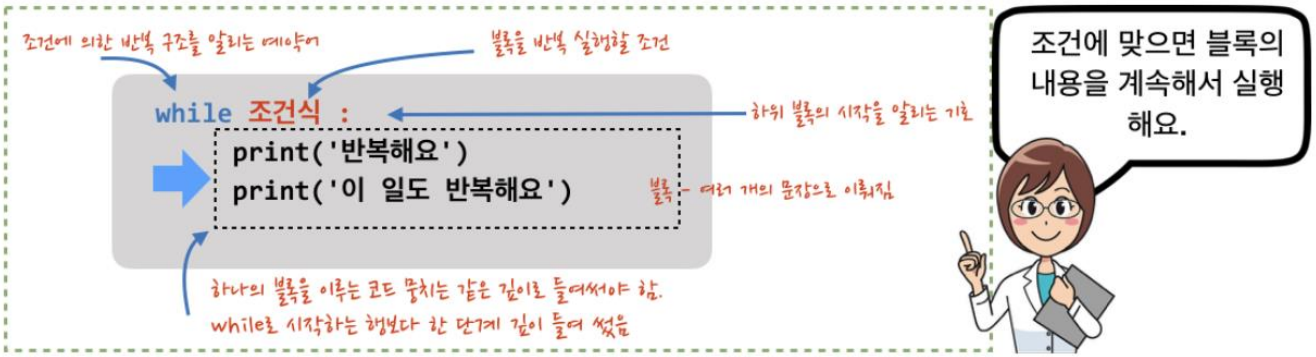
```
정수를 입력하시오: 10
10!은 3628800 이다.
```

```python
n = int(input("정수를 입력하시오: "))
fact = 1
for i in range(1, n+1):
    fact = fact * i

print(n, "!은", fact, "이다.")
```

while 문

- while 문 구조



while 문

어떤 경우에 반복되는 횟수를 알 수 없을까? 가장 대표적인 경우가 사용자로부터 어떤 값을 받아서 처리할 때이다. 사용자가 입력하는 값을 예측하기 힘들기 때문이다. 예를 들어서 사용자가 암호를 입력하고 프로그램에서 암호가 맞는지 체크한다고 하자.

우리가 작성할 코드는 사용자가 바른 암호를 입력할 때까지 질문을 계속한다. 암호가 "pythonisfun"이라고 가정하고 간단한 알고리즘을 생각해 보자. 다음과 같이 틀린 암호를 하나 생성해 두고, 사용자의 입력으로 대체한다. 그리고 이것이 옳은 암호인지를 체크해서 아닌 경우에는 이 일을 반복한다. 반복을 빠져나가면 로그인 성공을 표시해 준다.

1. 암호=""
2. 암호가 "pythonisfun"이 아니면 다음을 반복한다.
    - 사용자로부터 암호를 입력받는다.
3. "로그인 성공"을 출력한다.

원하는 결과

```
암호를 입력하시오: idontknow
암호를 입력하시오: 12345678
암호를 입력하시오: pythonisfun
** 로그인 성공 **
```

while 문

```
password = ""
while password != "pythonisfun":
    password = input("암호를 입력하시오: ")
print("** 로그인 성공 **")
```

while 문

- 1부터 10까지의 합을 계산한다면

```
count = 1
s = 0        # s는 누적하여 더한 값을 담을 변수로 0으로 초기화 함
while count <= 10 :
    s = s + count        # 매번 count 값을 s에 더함
    count = count + 1    # 매번 count 값을 1씩 증가시킴
print("합계는", s)
```

```
합계는 55
```

while 문

count	s	count <= 10	반복 여부
1	1	True	반복
2	1+2	True	반복
3	1+2+3	True	반복
4	1+2+3+4	True	반복
5	1+2+3+4+5	True	반복
6	1+2+3+4+5+6	True	반복
7	1+2+3+4+5+6+7	True	반복
8	1+2+3+4+5+6+7+8	True	반복
9	1+2+3+4+5+6+7+8+9	True	반복
10	1+2+3+4+5+6+7+8+9+10	True	반복
11	1+2+3+4+5+6+7+8+9+10	False	반복 중단!

사용자로부터 특정한 수를 입력받아 구구단의 특정 단을 while 반복문을 이용하여 출력하여 보자. 예를 들어 다음과 같이 9를 입력받을 경우 9*1, 9*2, 9*3, .., 9*9까지 9번 반복시켜 출력하면 될 것이다.

원하는 결과

```
원하는 단은: 9
9*1=9
9*2=18
9*3=27
9*4=36
9*5=45
9*6=54
9*7=63
9*8=72
9*9=81
```

```python
dan = int(input("원하는 단은: "))
i = 1

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

사용자가 입력한 숫자들을 더하는 프로그램을 작성해보자. 사용자가 yes라고 답한 동안에만 숫자를 입력받는다.

원하는 결과

```
숫자를 입력하시오: 10
계속?(yes/no): yes
숫자를 입력하시오: 20
계속?(yes/no): no
합계는 : 30
```

```
total = 0
answer = 'yes'
while answer == 'yes':
    number = int(input('숫자를 입력하시오: '))
    total = total + number
    answer = input('계속?(yes/no): ')

print('합계는 : ', total)
```

사용자가 답을 제시하면 프로그램은 자신이 저장한 정수와 비교하여 제시된 정수가 더 높은지 낮은지 만을 알려준다. 정수의 범위를 1부터 100까지로 한정하면 최대 7번이면 누구나 알아맞힐 수 있다. 정수의 범위를 1부터 1,000,000까지 확대하더라도 최대 20번이면 맞출 수 있다. 왜 그럴까? 이진 탐색의 원리 때문이다. 중간값과 한 번씩 비교할 때마다 탐색의 범위는 1/2로 대폭 줄어든다. 게임이 끝나면 몇 번 만에 맞추었는지도 함께 출력한다. 다음의 경우 87이 정답이지만 매번 임의로 생성된 수가 정답이 될 수 있음에 유의하자.

원하는 결과

```
1부터 100 사이의 숫자를 맞추시오
숫자를 입력하시오: 50
낮음!
숫자를 입력하시오: 86
낮음!
숫자를 입력하시오: 87
축하합니다. 총 시도횟수 = 3
```

```python
import random

tries = 0
guess = 0
answer = random.randint(1, 100)
print("1부터 100 사이의 숫자를 맞추시오")
while guess != answer:
    guess = int(input("숫자를 입력하시오: "))
    tries = tries + 1
    if guess < answer:
        print("낮음!")
    elif guess > answer:
        print("높음!")

print("축하합니다. 총 시도횟수=", tries)
```

달수는 새로 샌드위치 가게를 차렸다. 달수는 자신의 가게에서 제공가능한 빵, 고기, 야채, 소스의 조합을 통해서 만들 수 있는 모든 샌드위치의 종류를 출력하고 싶다. 달수네 가게의 빵 종류는 "호밀빵", "위트", "화이트"가 가능하며 고기로는 "미트볼", "소시지", "닭가슴살", 야채로는 "양상추", "토마토", "오이", 소스로는 "마요네즈", "허니 머스타드", "칠리" 등이 가능하다. 각 재료들은 한 가지씩만 선택이 가능하다고 하자. 가능한 조합은 어떻게 될까? 다음과 같이 모든 조합을 출력해 보자.

원하는 결과

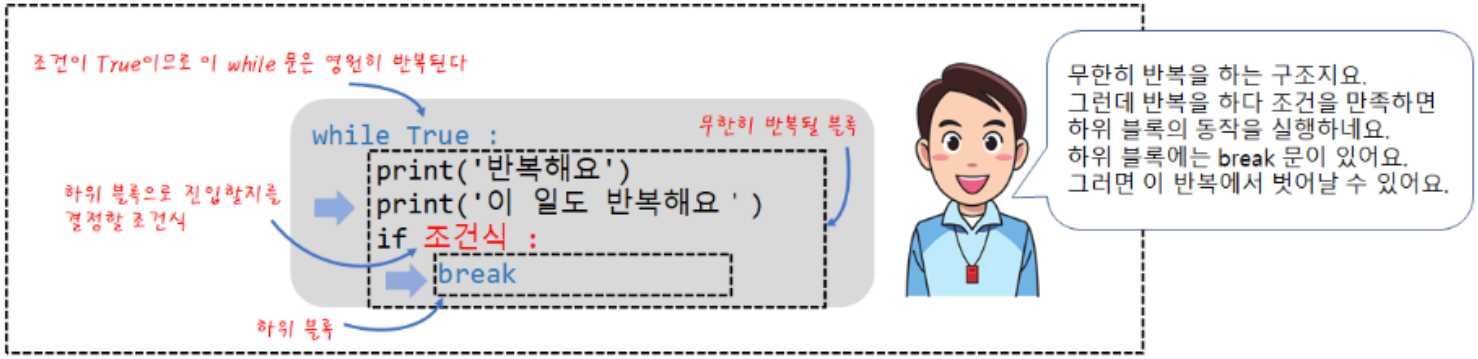
```
달수네 샌드위치 가게의 가능한 조합
호밀빵 + 미트볼 + 양상추 + 마요네즈
호밀빵 + 미트볼 + 양상추 + 허니 머스타드
호밀빵 + 미트볼 + 양상추 + 칠리
...
화이트 + 닭가슴살 + 오이 + 마요네즈
화이트 + 닭가슴살 + 오이 + 허니 머스타드
화이트 + 닭가슴살 + 오이 + 칠리
```

```python
breads = ["호밀빵", "위트", "화이트" ]
meats = ["미트볼", "소시지", "닭가슴살"]
vegis = ["양상추", "토마토", "오이"]
sauces = ["마요네즈", "허니 머스타드", "칠리"]

print('달수네 샌드위치 가게의 가능한 조합')
for b in breads:
    for m in meats:
        for v in vegis:
            for s in sauces:
                print(b + " + " + m + " + " + v + " + " + s)
```

무한 루프

- 무한 루프 구조


조건이 True이므로 이 while 문은 영원히 반복된다
while True :
print('반복해요')
print('이 일도 반복해요 ')
if 조건식 :
break
무한히 반복될 블록
하위 블록으로 진입할지를 결정할 조건식
하위 블록
무한히 반복을 하는 구조지요.
그런데 반복을 하다 조건을 만족하면
하위 블록의 동작을 실행하네요.
하위 블록에는 break 문이 있어요.
그러면 이 반복에서 벗어날 수 있어요.

무한 루프

- break 문장은 무한 루프를 빠져 나올 때 사용

```
while True:
    light = input('신호등 색상을 입력하시오:')
    if light == 'green':
        break
print('전진!!')
```

```
신호등 색상을 입력하시오: red
신호등 색상을 입력하시오: yellow
신호등 색상을 입력하시오: green
전진!!
```

무한 루프

도전문제 5.12

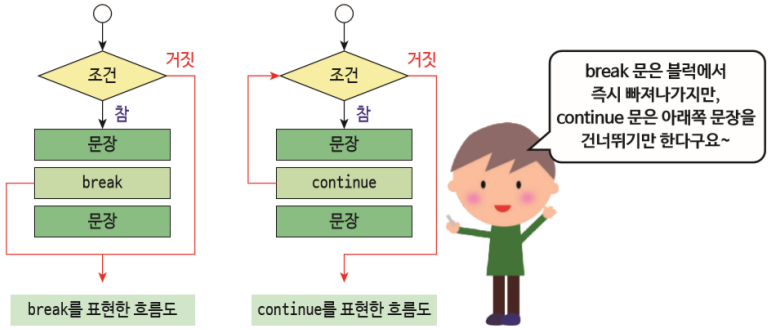
사용자로부터 하나의 단어를 입력받은 다음 이 단어에서 모음이 나타나기 전까지의 모든 자음을 출력하는 프로그램을 작성하여라. 예를 들어 다음과 같이 'programming'이 입력되면 'o'가 나타나기 이전인 pr만 출력하도록 하여라.

```
단어를 입력하세요 : programming
pr
```

무한 루프

```
st = 'I love Python Programming'  # 출력을 위한 문자열
for ch in st:
    if ch in ['a','e','i','o','u', 'A','E','I','O','U']:
        continue      # 모음일 경우 아래 출력을 건너뛴다
    print(ch, end='')
```

```
lv Pythn Prgrmmng
```



출력 포맷

- format() 메소드
  - 문자열 출력 형식을 결정

```
>>> '{} Python!'.format('Hello')
Hello Python!
>>> 'I like {} and {}'.format('Python', 'Java')
'I like Python and Java'
```

  - {}은 플레이스 홀더
  - format() 메소드의 인자로 들어오는 값의 출력되는 위치를 지정

출력 포맷

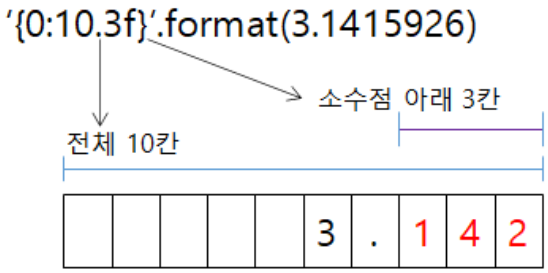
- 0, 1, 2와 같은 인덱스 활용 가능

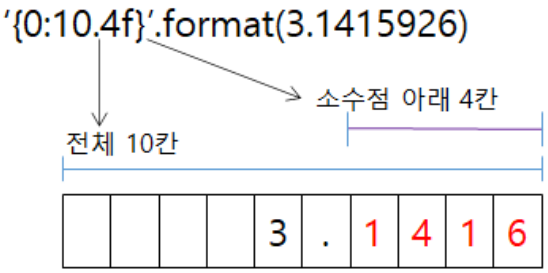
```
>>> 'I like {0} and {1}'.format('Python', 'Java')
'I like Python and Java'
>>> 'I like {1} and {0}'.format('Python', 'Java')
'I like Java and Python'
>>> 'I like {0}, {0}, {0}'.format('Python', 'Java')
'I like Python, Python, Python'
```

- 소수점 아래 둘째자리까지 숫자 출력
  - {0:.2f}
  - f는 부동소수점(floating point)를 의미
- {0:5.2f}, {0:6.2f}와 같은 형태도 가능

```
>>> '소수점 아래 두자리 정밀도 : {0:.2f}, 세자리 정밀도 {0:.3f}'.format(1/3)
소수점 아래 두자리 정밀도 : 0.33, 세자리 정밀도 0.333
>>> '전체 10칸을 차지하는 실수 : {0:10.3f}, {0:10.4f}'.format(3.1415926)
전체 10칸을 차지하는 실수 :      3.142,     3.1416
```

출력 포맷


'{0:10.3f}'.format(3.1415926)
소수점 아래 3칸
전체 10칸
3 . 1 4 2


'{0:10.4f}'.format(3.1415926)
소수점 아래 4칸
전체 10칸
3 . 1 4 1 6

출력 포맷

- 정수 출력도 {1:3d}, {1:4d}와 같이 차지할 칸 지정 가능

```
>>> for i in range(2, 11, 2):
...     print('{0:3d} {1:4d} {2:5d}'.format(i, i*i, i*i*i))
...
  2    4    8
  4   16   64
  6   36  216
  8   64  512
 10  100 1000
```