1. 구현 알고리즘
   - 두 수의 최대 공약수 구하기(gcd)
     - 작은 수를 큰 수로 나눈 나머지 값이 0이 될 때까지 진행
     - 만약 나머지가 0이면 마지막으로 나눈 값이 최대 공약수
   - 수 list에서 최대 공약수 구하기 (gcdVector)
     - 수 list의 갯수가 1개면 그 자체가 list의 최대 공약수
     - 만약 2개면 2개의 최대 공약수가 list의 최대 공약수
     - 만약 3개 이상이면 list에서의 한 개의 수와 나머지 list의 최대 공약수의 최대 공약수를 구한다.(재귀적 호출)
   - 두 수 사이의 소수 갯수 구하기(prime_number_between)
     - 작은 수를 큰 수가 될 때까지 증가시키면서
     - 작은 수가 소수인지 판별하고 맞으면 count를 증가시킨다.

2. 언어들 간의 차이
   - 변수 선언
     - c++/java : long a,b;
     - python : a = 10
   - 함수 선언
     - c++ : long long gcd(long long a, long long b){}
     - python : def gcd(a,b):
     - java : long gcd(long a, long b){}
   - 배열 선언 및 사용
     - c++ : vector 활용

```
vector<long long> numbers;
numbers[0];
numbers.pop_back();
```

     - python : list 활용

```
numbers[0]
numbers.pop(index)
```

     - java : vector 활용

```
Vector<Long> numbers = new Vector<Long>();
numbers.get(index);
numbers.remove(index);
```

   - 중복 값 제거 및 정렬
     - c++

```
numbers.erase(unique(numbers.begin(), numbers.end()), numbers.end());
sort(numbers.begin(), numbers.end());
```

     - python

```
numbers = list(set(numbers))
numbers.sort()
```

     - java

```
LinkedHashSet<Long> hashSet = new LinkedHashSet<Long>(numbers);
numbers.clear();
numbers.addAll(hashSet);
Collections.sort(numbers);
```

   - 시간 계산
     - c++ : 함수 포인터를 이용한 시간 계산

```
void myTimer(void (*fnc)()) {
        clock_t begin = clock();
        fnc();
        cout << "Total execution time using C++ is " << float(clock() - begin) /
    CLOCKS_PER_SEC << " seconds!\n";
}
```

- python : decorator 를 이용한 시간 계산

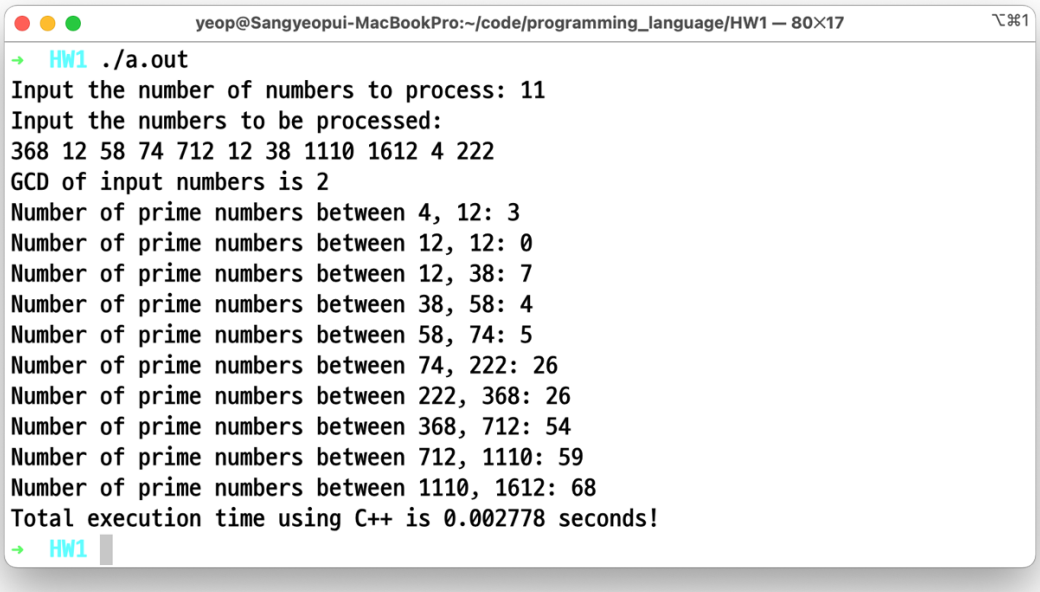
```
def timing(f):
    def wrap(*args, **kwars):
        ts = time()
        result = f(*args, **kwars)
        te = time()
        print("Total execution time using Python is ", te - ts, "Seconds!")
        return result
    return wrap
```

- java : solver 사이에 시간 계산

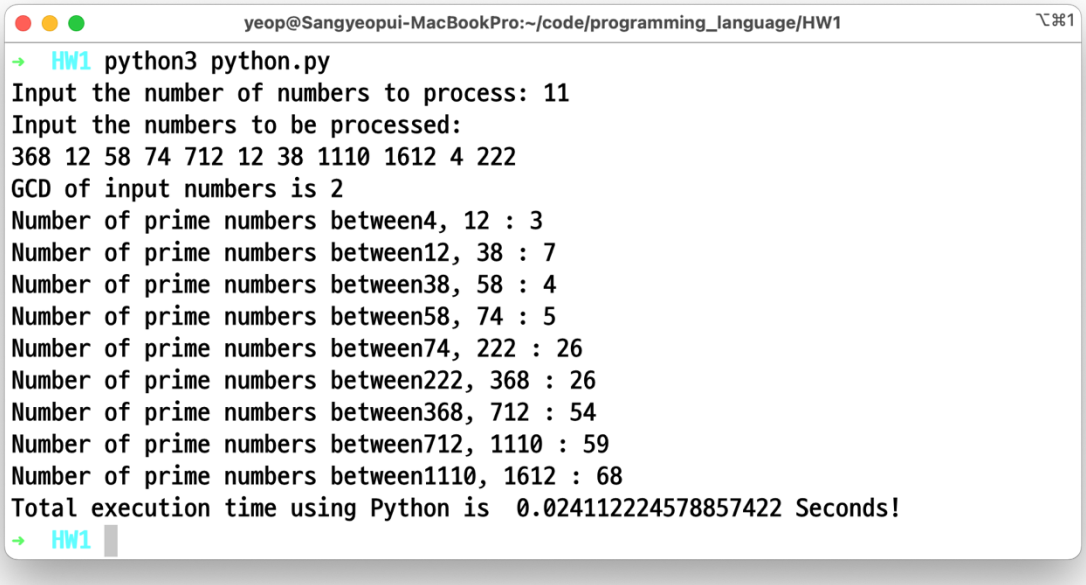
```
startTime = System.nanoTime();
 endTime = System.nanoTime();

 System.out.printf("Total execution time using Python is %f Seconds!\n", (endTime - startTime) /
(float) 1000000000);
```

3. 수행 결과

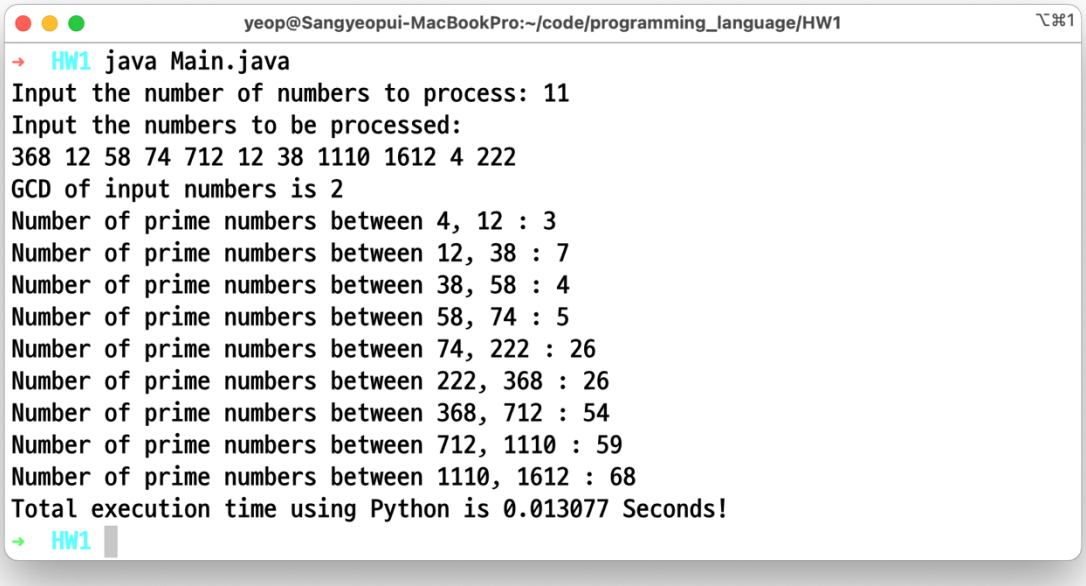
- c++



- python



- java



4. 느낀점
   - Readability(가독성)
     - python > c++ >> java
     - python은 변수의 형을 기술하지 않아도 되고, 선언도 단순히 대입연산자만 이용하면 되서 단순한 영문에 가까웠다. 또한 decorate 사용이 매우 간단해 좋았다.
     - c++은 익숙한 언어에 읽기가 매우 편하다는 생각이 들었다. 특히 python과 대비하여 main 함수에 수행할 함수들이 순서대로 나열되어 있다는 점이 읽기 쉽다는 생각이 들었다.
     - java는 Main 함수 하나 작성을 하기 위해서 class를 작성해야하며, 함수를 선언할때 또한 class에 종속적으로 이뤄져 있어 매우 귀찮다고 느껴졌다. 또한 Input/output 시 System class에서 가져와야 한다는 점이 가독성과 작성에 매우 큰 불편함을 느꼈다.
   - Writability(작성유용성)
     - python > c++ >> java
     - 모두들 Vector와 list와 같은 가변적인 배열을 제공하는 점에서 만족스러웠다.
     - python은 위에서 말한 것처럼 변수의 형을 기술하지 않아도 되는 점, 리스트 안에서 for문을 활용할 수 있는 등 여러 강점이 있지만 중괄호로 묶여 있지 않아 가끔 오류가 나는 점이 가장 큰 단점이었다.
     - c++은 역시 익숙하여 작성하기 가장 유용하였지만, 형 작성 및 함수 선언 순서, 그리고 decorator의 부재가 단점이었다.
     - java는 class 익숙하지 않아서 쓰기 많이 귀찮다고 느껴졌고, print나 scanner 등을 사용할때 매우 긴 함수를 써야되는 게 너무 쓰기 힘들었다.
   - Reliability(신뢰성)
     - c++ >= java >> python
     - c++이나 java에서는 형을 기술하여 잘못된 값이 입력되면 바로 실행을 종료시켜주지만 Python에서는 그런 기능이 없어 실행 중에 갑자기 오류가 난다.
     - 단, Exception을 일일히 적는다는 가정하에서는 Java >= Python > c++ 순으로 실행 중 오류처리를 잘 잡아주는 것은 물론 이상한 값만 제외하고 실행하는 등의 문을 쉽게 작성할 수 있다. 물론 c++도 if문을 활용해 처리할 수 있다.
   - Cost(비용; 여기선 시간을 중점으로 기술한다.)
     - c++(0.0027s) >> java(0.0130s) > python(0.0241s)
     - 위의 입력 이후 전처리부터 수행시간은 c++이 가장 짧았고, java가 c++의 약 5배 python이 약 10배 정도의 수행시간이 걸렸다. 이 점으로 미뤄보아 py

   - 나만의 결론
     - cost나 reliability를 고려하지 않아도 되는 단순 test나 prototyping의 경우 python이 이해하기도 작성하기도 쉬워서 무조건 해당 경우에서는 무조건 python이 유리한 것 같다.
     - 하지만 이외의 경우 class를 사용하지 않아도 main 함수나 기타 함수를 작성할 수 있으며 속도가 매우 우수할 뿐더러 최대한 코드를 줄일 수 있는 c++를 사용하는 것이 유리하다는 판단이 들었다.