LECTURE 04

USER DEFINED FUNCTION

Big-O Coding

Website: www.bigocoding.com

Giới thiệu

BT6 – TÌM SỐ NGUYÊN TỐ GẦN N NHẤT

- Cho một số nguyên dương n, hãy tìm số nguyên tố m gần nhất với n.
  - Nếu n là số nguyên tố thì m = n.
  - Giả sử có ba số a < b < c, số a được nói gần với b hơn so với c khi: $b - a < c - b$
  - Nếu có hai số cùng gần với n thì chọn số nhỏ hơn.
- Ví dụ:
  - Input: 5 → Output: 5
  - Input: 10 → Output: 11
  - Input: 4 → Output: 3

BT6 – Gợi ý

1. Đọc số nguyên dương n.
2. n1 = n, n2 = n.
3. Sử dụng vòng lặp, cho n1 từ n, chạy lùi dần, n1-=1.
   - Nếu n1 là số nguyên tố, thì dừng vòng lặp lại.
4. Sử dụng vòng lặp, cho n2 từ n, chạy tiến dần, n2+=1.
   - Nếu n2 là số nguyên tố, thì dừng vòng lặp lại.
5. Xét xem n-n1 vs n2-n, số nào nhỏ hơn, để in n1 hoặc n2.

BT6 – Lời giải (main only)

```
n = int(input())

n1 = n
n2 = n
found1 = False
found2 = False

while found1 == False:
    flag1 = True
    for i in range(2, n1):
        if n1 % i == 0:
            flag1 = False
    if(flag1 == True):
        found1 = True
    else:
        n1 -= 1
```

Kĩ thuật đặt cờ hiệu

BT6 – Lời giải (main only)

```
while found2 == False:
    flag2 = True
    for i in range(2, n2):
        if n2 % i == 0:
            flag2 = False
    if(flag2 == True):
        found2 = True
    else:
        n2 += 1
if n - n1 <= n2 - n:
    print(n1)
else:
    print(n2)
```

BT6 – Vấn đề 1

- Code quá dài, khó đọc.

  ➔ Có cách nào chia thành các đoạn nhỏ, dễ đọc?

BT6 – Vấn đề 2

- Đoạn code kiểm tra số nguyên tố lặp lại 2 lần:
  - Lần 1 để kiểm tra n1 có phải là số nguyên tố ko (found1 == True)

```
flag1 = True
for i in range(2, n1):
    if n1 % i == 0:
        flag1 = False
```

  - Lần 2 để kiểm tra n2 có phải là số nguyên tố ko (found2 == True)

```
flag2 = True
for i in range(2, n2):
    if n2 % i == 0:
        flag2 = False
```

BT6 – Vấn đề 2

- Do đó, sau khi đã code xong kiểm tra số nguyên tố cho n1.
    1. Ta phải tốn thời gian **code lại** phần kiểm tra số nguyên tố cho n2.
    2. Hoặc có thể **copy – paste**, nhưng phải nhớ chỉnh **sửa lại** các biến cho phù hợp.

BT6 – Vấn đề 2

```
flag1 = True
for i in range(2, n1):
    if n1 % i == 0:
        flag1 = False
```

```
flag2 = True
for i in range(2, n2):
    if n1 % i == 0:
        flag2 = False
```

BT6 – Vấn đề 2

- Khi sửa code ở chỗ-giống-nhau-1 thì cũng phải nhớ sửa code ở chỗ-giống-nhau-2.
- VD: nhận thấy rằng vòng lặp chỉ cần kiểm tra **từ 2 đến sqrt(n1)** là đủ.

BT6 – Vấn đề 2

```
    flag1 = True
    for i in range(2, int(n1 ** 0.5) + 1):
        if n1 % i == 0:
            flag1 = False
```

```
    flag2 = True
    for i in range(2, int(n2 ** 0.5) + 1):
        if n2 % i == 0:
            flag2 = False
```

Giải pháp tốt hơn: Định nghĩa hàm isPrime()

Cài đặt và sử dụng hàm

Hàm là gì?

- Hàm là một đoạn chương trình, gồm nhiều dòng code, được nhóm lại với nhau và được đặt tên.
- Thông qua tên hàm, developer gọi thực hiện hàm đó nhiều lần khác nhau với các tham số khác nhau.
- Khai báo (declare) và cài đặt (define): 1 lần
- Gọi sử dụng (call, invoke) nhiều lần.

Công dụng của hàm

- Giảm thiểu những đoạn code trùng lắp, giúp dễ chỉnh sửa về sau.
- Chia chương trình thành các hàm con (subtask) dễ code, dễ đọc.

User-defined function

- Một số hàm sẵn có:
  - C++: sin(), sqrt(), pow(), log(), round()…
  - Java: Math.sin(), Math.sqrt(), Math.pow(), Math.log(), Math.round()…
  - Python: input(), print(), sqrt(), sin(), log(), round()…
- Trong bài này, chúng ta sẽ họ **viết thêm các hàm mới**.

Các bước viết hàm

- Xác định:
  - Tên hàm.
  - Đầu vào.
  - Đầu ra.
  - Các xử lí trong hàm.
- Cài đặt:
  - Cài đặt xử lí của hàm.
  - Gọi hàm.

Khai báo hàm (function declaration / prototype) & Cài đặt hàm (function definition)

```
return_type nameOfFunction(type arg1, type arg2);
```

```
return_type nameOfFunction(type arg1, type arg2){
    Các dòng code xử lý ;
    Dùng lệnh return để trả về kết quả;
}
```

C++: khai báo có thể tách rời với cài đặt hàm

```
static return_type nameOfFunction(type arg1, type arg2){
    Các dòng code xử lý ;
    Dùng lệnh return để trả về kết quả;
}
```

Java, Python: khai báo và cài đặt hàm viết chung

```
def nameOfFunction(arg1, arg2):
    Các dòng code xử lý
    Dùng lệnh return để trả về kết quả
```

Ý nghĩa

- **nameOfFunction**: tên hàm, đặt theo qui tắc đặt tên biến, nên dùng động từ (khác với biến, nên dùng danh từ)
- **arg1, arg2**: các tham số đầu vào
- **return**: kết thúc 1 hàm, trả về giá trị tính toán được.
  - C++, Java: phải khai báo kiểu dữ liệu của giá trị trả về (int, float, Product, void…)
  - Python: ko cần khai báo kiểu dữ liệu của giá trị trả về.

VD1: Hàm không đầu vào, không đầu ra (C++)

```
Tên hàm: add
Đầu vào: (không có)
Đầu ra: (không có)
Xử lí: cộng 2 số nguyên
```

```cpp
// Khai báo hàm
void add();
```

C++: khai báo hàm phải đặt trước lời gọi hàm

```cpp
// Cài đặt hàm
void add(){
    int num1, num2, num3;
    cin >> num1 >> num2;
    num3 = num1 + num2;
    cout << num3;
}
```

```cpp
// Gọi hàm
int main(){
    add();
    return 0;
}
```

VD1: Hàm không đầu vào, không đầu ra (Java)

```
Tên hàm: add
Đầu vào: (không có)
Đầu ra: (không có)
Xử lí: cộng 2 số nguyên
```

```java
// Cài đặt hàm
public static void add(){
    int num1, num2, num3;
    Scanner sc = new Scanner(System.in);
    num1 = sc.nextInt();
    num2 = sc.nextInt();
    num3 = num1 + num2;
    System.out.print(num3);
}
```

Java: gộp chung khai báo hàm và cài đặt hàm

```java
// Gọi hàm
public static void main(String[] args){
    add();
}
```

VD1: Hàm không đầu vào, không đầu ra (Python)

```
Tên hàm: add
Đầu vào: (không có)
Đầu ra: (không có)
Xử lí: cộng 2 số nguyên
```

```python
# Cài đặt hàm
def add():
    num1 = int(input())
    num2 = int(input())
    num3 = num1 + num2
    print(num3)
```

Python: khai báo hàm phải đặt trước lời gọi hàm

```python
# Gọi hàm
add()
```

VD2: Hàm có đầu vào, không đầu ra (C++)

```
Tên hàm: add
Đầu vào: 2 số nguyên
Đầu ra: (không có)
Xử lí: cộng 2 số nguyên
```

```cpp
// Khai báo hàm
void add(int num1, int num2);
```

```cpp
// Cài đặt hàm
void add(int num1, int num2){
    int num3;
    num3 = num1 + num2;
    cout << num3;
}
```

Tên biến truyền vào vs tên tham số ko cần giống nhau, miễn là cùng kiểu dữ liệu: (a vs num1: int; b vs num2: int)

```cpp
// Gọi hàm
int main(){
    int a, b;
    cin >> a >> b;
    add(a, b);
    return 0;
}
```

VD2: Hàm có đầu vào, không đầu ra (Java)

```
Tên hàm: add
Đầu vào: 2 số nguyên
Đầu ra: (không có)
Xử lí: cộng 2 số nguyên
```

```java
// Cài đặt hàm
public static void add(int num1, int num2){
    int num3;
    num3 = num1 + num2;
    System.out.print(num3);
}
```

Tên biến truyền vào vs tên tham số ko cần giống nhau, miễn là cùng kiểu dữ liệu: (a vs num1: int; b vs num2: int)

```java
// Gọi hàm
public static void main(String[] args){
    int a, b;
    Scanner sc = new Scanner(System.in);
    a = sc.nextInt();
    b = sc.nextInt();
    add(a, b);
}
```

VD2: Hàm có đầu vào, không đầu ra (Python)

```
Tên hàm: add
Đầu vào: 2 số nguyên
Đầu ra: (không có)
Xử lí: cộng 2 số nguyên
```

```python
# Cài đặt hàm
def add(num1, num2):
    num3 = num1 + num2
    print(num3)
```

Tên biến truyền vào vs tên tham số ko cần giống nhau

```python
# Gọi hàm
a = int(input())
b = int(input())
add(a, b)
```

VD3: Hàm có đầu vào, có đầu ra (C++)

```
Tên hàm: add
Đầu vào: 2 số nguyên
Đầu ra: tổng
Xử lí: cộng 2 số nguyên
```

```cpp
// Khai báo hàm
int add(int num1, int num2);
```

```cpp
// Cài đặt hàm
int add(int num1, int num2){
    int num3;
    num3 = num1 + num2;
    return num3;
}
```

```cpp
// Gọi hàm
int main(){
    int a, b, c;
    cin >> a >> b;
    c = add(a, b);
    cout << c;
    return 0;
}
```

VD3: Hàm có đầu vào, có đầu ra (Java)

```
Tên hàm: add
Đầu vào: 2 số nguyên
Đầu ra: tổng
Xử lí: cộng 2 số nguyên
```

```java
// Cài đặt hàm
public static int add(int num1, int num2){
    int num3;
    num3 = num1 + num2;
    return num3;
}
```

```java
// Gọi hàm
public static void main(String[] args){
    int a, b, c;
    Scanner sc = new Scanner(System.in);
    a = sc.nextInt();
    b = sc.nextInt();
    c = add(a, b);
    System.out.print(c);
}
```

VD3: Hàm có đầu vào, có đầu ra (Python)

```
Tên hàm: add
Đầu vào: 2 số nguyên
Đầu ra: tổng
Xử lí: cộng 2 số nguyên
```

```python
# Cài đặt hàm
def add(num1, num2):
    num3 = num1 + num2
    return num3
```

```python
# Gọi hàm
a = int(input())
b = int(input())
c = add(a, b)
print(c)
```

VD4: Hàm isPrime()

```
bool isPrime(int n){
   if(n < 2)
      return false;
   for(int i = 2; i < int(sqrt((double)n) + 1); i++){
      if(n % i == 0)
         return false;
   }
   return true;
}
```

```
public static boolean isPrime(int n){
    if(n < 2)
        return false;
    for(int i = 2; i < int(Math.sqrt(n) + 1); i++){
        if(n % i == 0)
            return false;
    }
    return true;
}
```

```
def isPrime(n):
    if(n < 2):
        return False
    for i in range(2, int(n ** 0.5) + 1):
        if(n % i == 0):
            return False
    return True
```

Một số lưu ý

Lưu ý 1: Biến cục bộ (local variable)

- Mỗi biến được tạo ra trong một khối lệnh, chỉ có tác dụng cục bộ trong hàm đó.

VD: Biến cục bộ (C++)

```cpp
// Cài đặt hàm
int add(int num1, int num2){
    int num3 = num1 + num2;
    return num3;
}
```

```cpp
// Gọi hàm
int main() {
    int a = 4;
    int b = 5;
    int c = add(a, b);
    cout << num1 << endl;
    cout << num2 << endl;
    cout << num3 << endl;
    return 0;
}
```

```
# Compiled-time error
error: use of undeclared identifier 'num1'
error: use of undeclared identifier 'num2'
error: use of undeclared identifier 'num3'
```

VD: Biến cục bộ (Java)

```java
static int add(int num1, int num2){
    int num3 = num1 + num2;
    return num3;
}
```

```java
public static void main(String[] args) {
    int a = 4;
    int b = 5;
    int c = add (a, b);
    System.out.println(num1);
    System.out.println(num2);
    System.out.println(num3);
}
```

```
# Compiled-time error
java: cannot find symbol num1
java: cannot find symbol num2
java: cannot find symbol num3
```

VD: Biến cục bộ (Python)

```
# Cài đặt hàm
def add(num1, num2):
    num3 = num1 + num2
    return num3
```

```
# Gọi hàm
a = 4
b = 5
c = add(a, b)
print(num1)
print(num2)
print(num3)
```

```
# Run-time error
NameError: name 'num1' is not defined
NameError: name 'num2' is not defined
NameError: name 'num3' is not defined
```

Lưu ý 2: Biến toàn cục (global variable)

- Biến toàn cục là biến khai báo ở bên ngoài các hàm.

Java không có biến toàn cục

```
# Tạo biến toàn cục
int x = 8;

# Cài đặt hàm
void testGlobal(){
    cout << x;
}

# Gọi sử dụng hàm
int main(){
    testGlobal();
    return 0;
}
```

```
# Tạo biến toàn cục
x = 8

# Cài đặt hàm
def testGlobal():
    print(x)


# Gọi sử dụng hàm

testGlobal()
```

SHOULD NOT

Thay đổi giá trị toàn cục trong hàm

```cpp
int x = 8;

void testGlobal(){

    x = 10;
}

int main(){
    cout << x << endl;
    testGlobal();
    cout << x << endl;
    return 0;
}
```

```python
x = 8

def testGlobal():
    global x
    x = 10



print(x)
testGlobal()
print(x)
```

Lưu ý 3: Tham số mặc định (default argument)

```cpp
#include <math.h>
double powerOf(double x, int y = 2) {
    return pow(x, y);
}

int main() {
    cout << powerOf(3) << endl;
    cout << powerOf(3, 4) << endl;
}
```

Java không có tham số mặc định

```python
def powerOf(x, y = 2):
    return x ** y

print(powerOf(3))
print(powerOf(3, 4))
```

Lưu ý 4: overloading functions

- Overloading functions là các hàm có cùng tên, nhưng khác nhau về số lượng tham số đầu vào, hoặc khác nhau về kiểu dữ liệu của tham số đầu vào.

Ví dụ (C++)

```cpp
int add(int x);
int add(int x, int y);
double add(int x, double y);
```

```cpp
int main() {
    int x = 3;
    int y1 = 5;
    double y2 = 3.14;
    cout << add(x) << endl;
    cout << add(x, y1) << endl;
    cout << add(x, y2) << endl;
    return 0;
}
```

```cpp
int add(int x);
int add(int y); // error
double add(int x); // error
```

```cpp
int add(int x){
    return x + 1;
}
int add(int x, int y){
    return x*10 + y;
}
double add(int x, double y){
    return x + y;
}
```

Ví dụ (Java)

```java
public static void main (String[] args)
{
    int x = 3;
    int y1 = 5;
    double y2 = 3.14;

    System.out.println(add(x));
    System.out.println(add(x, y1));
    System.out.println(add(x, y2));
}
```

```java
public static int add(int x){
    return x + 1;
}

public static int add(int x, int y){
    return x*10 + y;
}

public static double add(int x, double y){
    return x + y;
}
```

```java
public static int add(int x){
    return x + 1;
}

// error
public static int add(int y){
    return y + 2;
}

// error
public static double add(int
x){
    return x * 1.0 + 1;
}
```

Ví dụ (Python)

- Python không hỗ trợ overloading functions nhưng C++ và Java.
- Có thể sử dụng default parameters.

```
def add(x, y = 0, z = 0):
    return x + y + z

x = 3
y = 5
z = 7

print(add(x))
print(add(x, y))
print(add(x, y, z))
```

Lưu ý 5: Chia file (C++)

Big-O Green

```cpp
// Function.h
#ifndef _FUNCTION_H_
#define _FUNCTION_H_

#include <iostream>
#include <math.h>
using namespace std;

bool isPrime(int n);

#endif
```

```cpp
// Solution.cpp
#include "Function.h"
int main() {
    int n;
    cin >> n;
    bool ans = isPrime(n);
    if(ans)
        cout << "YES" << endl;
    else
        cout << "NO" << endl;
    return 0;
}
```

```cpp
// Function.cpp
#include "Function.h"

bool isPrime(int n){
    if(n < 2)
        return false;
    for(int i = 2; i < sqrt((double)n) + 1; i++){
        if(n % i == 0)
            return false;
    }
    return true;
}
```

Lưu ý 5: Chia file (Java)

Big-O Green

```java
// Utilities.java
public class Utilities {
    public static boolean isPrime(int n){
        if(n < 2)
            return false;
        for(int i = 2; i < Math.sqrt(n) + 1; i++){
            if(n % i == 0)
                return false;
        }
        return true;
    }
}
```

```java
// L01P01.java
import java.util.Scanner;
class Solution
{
    public static void main (String[] args)
    {
        int n;
        Scanner sc = new Scanner(System.in);
        n = sc.nextInt();
        boolean ans = Utilities.isPrime(n);
        if(ans)
            System.out.println("YES");
        else
            System.out.println("NO");
    }
}
```

Lưu ý 5: Chia file (Python)

```python
# Utilities.py
def isPrime(n):
    if(n < 2):
        return False
    for i in range(2, int(n ** 0.5) + 1):
        if(n % i == 0):
            return False
    return True
```

```python
#Solution.py
from Utilities import isPrime

n = int(input())
ans = isPrime(n)
if(ans):
    print("YES")
else:
    print("NO")
```

Lưu ý 6: Trả về nhiều giá trị trong hàm

- C++, Java: **không cho phép** 1 hàm trả về nhiều giá trị tại một thời điểm.
  - C++: Phải sử dụng tham số dạng tham chiếu (pass by reference).
  - Java: phải đóng gói vào 1 class.
- Python **cho phép** 1 hàm trả về nhiều giá trị tại 1 thời điểm.

VD: Trả về 2 giá trị (C++)

```
void square(int x,int y, int &a, int &b) {
    a = x*x;
    b = y*y;
}

int main(){
    int xsq = 0;
    int ysq = 0;
    square(2, 3, xsq, ysq);
    cout << xsq << endl;
    cout << ysq << endl;
}
```

VD: Trả về 2 giá trị (Java)

```
class SquareValue{
    public int xsq;
    public int ysq;
}
```

```
static SquareValue square(int x, int y){
    SquareValue sq = new SquareValue();
    sq.xsq = x * x;
    sq.ysq = y * y;
    return sq;
}
```

```
public static void main(String[] args) {
    SquareValue sq = square(2, 3);
    System.out.println(sq.xsq);
    System.out.println(sq.ysq);
}
```

VD: Trả về 2 giá trị (Python)

```
def square(x,y):
    return x*x, y*y

xsq, ysq = square(2,3)
print(xsq)
print(ysq)
```

Lưu ý 7: Pass by value vs Pass by reference

- **Pass by value (tham trị):** thay đổi giá trị của tham số bên trong hàm, nhưng **không ảnh hưởng** đến giá trị tham số ở **bên ngoài** hàm.

- **Pass by reference (tham chiếu):** thay đổi giá trị của tham số bên trong hàm, **ảnh hưởng** đến giá trị tham số ở **bên ngoài** hàm.

Lưu ý 7: Pass by value vs Pass by reference

- C++:
  - Tham trị: int x;
  - Tham chiếu: int &x;
- Java, Python:
  - Tham trị: các kiểu dữ liệu cơ bản int, float, Boolean… → primitive type
  - Tham chiếu: các kiểu dữ liệu khác string, list, Product… → referenced type

Ví dụ

```
void swap(int x, int y){
    int temp = x;
    x = y;
    y = temp;
    cout << "Inside swap(): " << x
         << " " << y << endl;
}

int main(){
    int a = 3, b = 5;
    cout << "Before swap(): " << a
         << " " << b << endl;
    swap(a, b);
    cout << "After swap(): " << a
         << " " << b << endl;
    return 0;
}

Before swap(): 3 5
Inside swap(): 5 3
After swap(): 3 5
```

```
void swap(int &x, int &y){
    int temp = x;
    x = y;
    y = temp;
    cout << "Inside swap(): " << x
         << " " << y << endl;
}

int main(){
    int a = 3, b = 5;
    cout << "Before swap(): " << a
         << " " << b << endl;
    swap(a, b);
    cout << "After swap(): " << a
         << " " << b << endl;
    return 0;
}

Before swap(): 3 5
Inside swap(): 5 3
After swap(): 5 3
```

Bài tập

Cài đặt ít nhất 1 hàm cho mỗi bài tập

BT1 – TỔNG SỐ NGUYÊN TỐ

- Hãy tính tổng các số nguyên tố nhỏ hơn n.

- Gợi ý:
  - Cài đặt hàm isPrime() để kiểm tra số n có phải là số nguyên tố hay không?

BT2 – TÍNH TỔNG

- Viết chương trình tính tổng S theo công thức.

$$S = 1^2 + 2^2 + 3^2 + \cdots + n^2$$

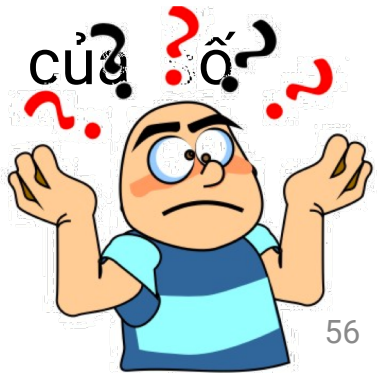
- Gợi ý:
  - Hàm sum(): tính S(n) theo công thức.

BT3 – RÚT GỌN PHÂN SỐ

- Cho phân số a/b với a và b là hai số nguyên dương.
- Hãy rút gọn phân số đã cho.
- Gợi ý:
  - Hàm GCD(): tìm ước chung lớn nhất của 2 số nguyên dương a, b.
  - GCD: Greatest Common Divisor.

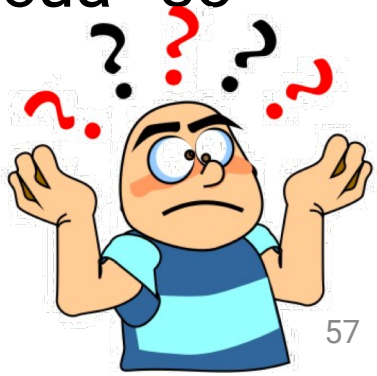
BT4 – ĐẾM SỐ LƯỢNG CHỮ SỐ

- Cho một số nguyên dương N, hãy cho biết số N có bao nhiêu chữ số.
- Ví dụ:
  - Input: 2018
  - Output: 4
- Gợi ý:
  - Hàm countDigit(): đếm số lượng chữ số của số nguyên dương n.



BT5 – CHỮ SỐ LỚN NHẤT

- Cho một số nguyên dương N, hãy cho biết chữ số lớn nhất của số N.
- Ví dụ:
  - Input: 8201
  - Output: 8
- Hàm maxDigit(): tìm chữ số lớn nhất của số nguyên dương n.



Hỏi đáp

Gợi ý

BT1 – Gợi ý xử lí chính

1. Đọc số nguyên dương n.
2. Khởi tạo ans = 0.
3. Sử dụng vòng lặp cho i chạy từ 1 đến n-1.
    1. Nếu isPrime(i) == true, cộng dồn i vào ans.
4. In kết quả.

BT1 – Gợi ý hàm isPrime()

- Tên hàm isPrime().
- Đầu vào: số nguyên n.
- Đầu ra: n có phải là số nguyên tố hay ko True/False.
- Xử lí: **kĩ thuật đặt cờ hiệu**
    1. Khởi tạo cờ hiệu: **flag = True**
    2. Nếu n < 2, flag = False. (số nhỏ hơn 2 ko phải là nguyên tố)
    3. Ngược lại,
        - Sử dụng vòng lặp, i chạy từ 2 đến n-1.
            - Nếu n chia hết cho i, n ko phải là số nguyên tố, **flag = False**.
    4. return flag

BT1 – Minh họa n = 6

6

n = 6
ans = 0
i = 1

1 < 6: True
isPrime(1): False
i = 1 + 1 = 2

2 < 6: True
isPrime(2): True
ans = 0 + 2 = 2
i = 2 + 1 = 3

3 < 6: True
isPrime(3): True
ans = 2 + 3 = 5
i = 3 + 1 = 4

4 < 6: True
isPrime(4): False
i = 4 + 1 = 5

5 < 6: True
isPrime(5): True
ans = 5 + 5 = 10
i = 5 + 1 = 6

6 < 6: False

BT2 – Gợi ý xử lí chính

1. Đọc vào số nguyên dương n.
2. ans = sumPower(n).
3. In ans.

BT2 – Gợi ý hàm sumPower()

- Đầu vào: số nguyên dương n.
- Đầu ra: tổng.
- Xử lí: **kĩ thuật tính tổng**
    1. Khởi tạo biến kết quả: **ans = 0**.
    2. Sử dụng vòng lặp for, start = ?, stop = ?, step = ?
        - **ans += i * i**
    3. return ans.

BT2 – Minh họa sumPower(5)

ans = 0
i = 1

1 < 6: True
ans = ans + i^2 = 0 + 1^2 = 1
i = i + 1 = 1 + 1 = 2

2 < 6: True
ans = ans + i^2 = 1 + 2^2 = 5
i = i + 1 = 2 + 1 = 3

3 < 6: True
ans = ans + i^2 = 5 + 3^2 = 14
i = i + 1 = 3 + 1 = 4

4 < 6: True
ans = ans + i^2 = 14 + 4^2 = 30
i = i + 1 = 4 + 1 = 5

5 < 6: True
ans = ans + i^2 = 30 + 5^2 = 55
i = i + 1 = 5 + 1 = 6

6 < 6: **False**
return 55

BT3 – Gợi ý xử lí chính

1. Đọc phân số, lưu vào 2 biến numerator, denominator.
2. x = GCD(abs(numerator), abs(denominator)
3. Chia numerator, denominator cho x.
4. In phân số đã được rút gọn.

BT3 – Gợi ý hàm GCD()

- Đầu vào: 2 số nguyên dương a, b.
- Đầu ra: ước chung lớn nhất.
- Xử lí: sử dụng giải thuật Euclidean
    1. Sử dụng vòng lặp b != 0.
        a. r = a % b
        b. a = b
        c. b = r
    2. return a.

BT3 – Minh họa phân số 24/45

numerator = 24
denominator = 45

x = GCD(24, 45) = 3
numerator = 24 / 3 = 8
denominator = 45 / 3 = 15

# Hàm GCD(24, 45)
a = 24
b = 45

b = 45 != 0: True
r = a % b = 24 % 45 = 24
a = b = 45
b = r = 24

b = 24 != 0: True
r = a % b = 45 % 24 = 21
a = b = 24
b = r = 21

b = 21 != 0: True
r = a % b = 24 % 21 = 3
a = b = 21
b = r = 3

b = 3 != 0: True
r = a % b = 21 % 3 = 0
a = b = 3
b = r = 0

b = 0 == 0: True
return 3

BT4 – Gợi ý xử lí chính

1. Đọc vào số nguyên dương n.
2. Gọi hàm countDigit(n), lưu kết quả vào ans.
3. In kết quả.

BT4 – Gợi ý hàm countDigit()

- Đầu vào: số nguyên dương n.
- Đầu ra: số lượng chữ số.
- Xử lí: **kĩ thuật đếm**
    1. Khởi tạo biến đếm: **count = 0**.
    2. Sử dụng vòng lặp while(n>0).
        a. Sử dụng phép toán % 10 để tách chữ số đơn vị của n: x = n % 10.
        b. Tăng biến count.
        c. Giảm n đi 10 lần, 2018 → 201.
    3. return count.

BT4 – Minh họa countDigit(2018)

n = 2018
count = 0

n = 2018 > 0: True
count = count + 1 = 0 + 1 = 1
n = n // 10 = 2018 // 10 = 201

n = 201 > 0: True
count = count + 1 = 1 + 1 = 2
n = n // 10 = 201 // 10 = 20

n = 20 > 0: True
count = count + 1 = 2 + 1 = 3
n = n // 10 = 20 // 10 = 2

n = 2 > 0: True
count = count + 1 = 3 + 1 = 4
n = n // 10 = 2 // 10 = 0

n = 0 > 0: False
return 4

BT5 – Gợi ý xử lí chính

1. Đọc vào số nguyên dương n.
2. Gọi hàm maxDigit(n), lưu kết quả vào ans.
3. In kết quả.

BT5 – Gợi ý hàm maxDigit()

- Đầu vào: số nguyên dương n.
- Đầu ra: chữ số lớn nhất.
- Xử lí: **kĩ thuật lính canh**
  1. Khởi tạo lính canh: **ans = -1**.
  2. Sử dụng vòng lặp while(n>0).
     a. Sử dụng phép toán % 10 để tách chữ số đơn vị của n: x = n % 10.
     b. Nếu chữ số hiện tại > chữ số lớn nhất: x > ans, cập nhật lính canh **ans = x**.
     c. Giảm n đi 10 lần, 2018 → 201.
  3. return ans.

BT5 – Minh họa maxDigit(8201)

n = 8201
ans = -1

n = 8201 > 0: True
x = n % 10 = 8201 % 10 = 1
x = 1 > ans = -1: True, ans = 1
n = n // 10 = 8201 // 10 = 820

n = 820 > 0: True
x = n % 10 = 820 % 10 = 0
x = 0 > ans = 1: False
n = n // 10 = 820 // 10 = 82

n = 82 > 0: True
x = n % 10 = 82 % 10 = 2
x = 2 > ans = 1: True, ans = 2
n = n // 10 = 82 // 10 = 8

n = 8 > 0: True
x = n % 10 = 8 % 10 = 8
x = 8 > ans = 2: True, ans = 8
n = n // 10 = 8 // 10 = 0

n = 0 > 0: False
return 8