Amazing Numbers

Stage 1/8: Buzz numbers

Description

Integers can be even or odd. All numbers that end with 0, 2, 4, 6, or 8 are even; the rest are odd. However, numbers have many other properties, you will learn about some of them in this project.

First, let's talk about **Buzz numbers**. They are numbers that are either divisible by 7 or end with 7. For example, the number 14 is a buzz number, since it is divisible by 7 without a remainder; the number 17 ends with 7, so it is also a buzz number. However, the number 75 is not a Buzz number, since it is neither divisible by 7 nor does it end with 7. The number 7 is a Buzz number too.

Your task at this stage is to write a program that prints the natural number parity and determines whether this number is a Buzz. The program should print why this number is a Buzz number.

Do you know how to determine whether a number is divisible by 7 or not? We have a great tip:

1. Remove the last digit;
2. Multiply the removed digit by 2 and subtract it from the remaining number.
3. If the result of the subtraction can be divided by 7, then the initial number is divisible by 7. You can apply the whole sequence multiple times until you get a relatively small/easy-to-interpret subtraction result.

For example, take 196. We remove the last digit and get 19. We subtract 12 (the removed digit multiplied by 2) to get 7. Since the last left digit is 7, the number is multiple of 7. So 196 is a Buzz number!

You can use any solution you want, the tests won't check it.

Objectives

Your program should check whether the given natural number is a buzz number.

1. Ask a user to enter a natural number.
2. If the number is not natural, print an error message.
3. Calculate and print the parity of the number.
4. Check whether is the number is a Buzz number and print the explanation.
5. Finish the program after printing the message.

Natural numbers are whole numbers starting from 1.

Explanations:

```
... is neither divisible by 7 nor does it end with 7
```

```
... is divisible by 7
```

```
... ends with 7
```

```
... is divisible by 7 and ends with 7
```

Error message:

```
This number is not natural!
```

Examples

The greater-than symbol followed by a space (`> `) represents the user input. Note that it's not part of the input.

Example 1:

```
Enter a natural number:
> 13
This number is Odd.
It is not a Buzz number.
Explanation:
13 is neither divisible by 7 nor does it end with 7.
```

Example 2:

```
Enter a natural number:
> 14
This number is Even.
It is a Buzz number.
Explanation:
14 is divisible by 7.
```

Example 3:

```
Enter a natural number:
> 17
This number is Odd.
It is a Buzz number.
Explanation:
17 ends with 7.
```

Example 4:

```
Enter a natural number:
> 7
This number is Odd.
It is a Buzz number.
Explanation:
7 is divisible by 7 and ends with 7.
```

Example 5:

```
Enter a natural number:
-76
This number is not natural!
```

Stage 2/8: Duck numbers

Description

Your next task is to determine whether a number is a **Duck number**. A Duck number is a positive number that contains zeroes. For example, 321**0**, 8**0**5**0**896, 7**0**7**0**9 are Duck numbers. Note that a number with a leading 0 is not a Duck number. So, numbers like 035 or 0212 are not Duck numbers. Although, 012**0**3 is a Duck, since it has a trailing 0.

In this stage, we need to simplify the way we display the information. We already have several properties that we need to show; we are going to add new features to our program in each stage. From now on, the card should follow this notation:

```
Properties of {number}
{property}: true/false
{property}: true/false
...
{property}: true/false
```

In this stage, the properties are `even`, `odd`, `buzz` and `duck`. For `14`, the program output should look like this:

```
Properties of 14
        even: true
         odd: false
        buzz: true
        duck: false
```

The property order, indentation, and spaces are not checked by the tests. The tests are also case-insensitive.

Objectives

Your program should print the properties of a natural number. In this stage, your program should:

1. Ask a user to enter a natural number;
2. If the number is not natural, the program should print an error message;
3. If the number is natural, the program should indicate the properties of the number;
4. Finish the program after printing the message.

Examples

The greater-than symbol followed by a space (`> `) represents the user input. Note that it's not part of the input.

Example 1:

```
Enter a natural number:
> -7
This number is not natural!
```

Example 2:

```
Enter a natural number:
> 15
Properties of 15
        even: false
         odd: true
        buzz: false
        duck: false
```

Example 3:

```
Enter a natural number:
> 14
Properties of 14
        even: true
         odd: false
        buzz: true
        duck: false
```

Example 4:

```
Enter a natural number:
> 102
Properties of 102
        even: true
         odd: false
        buzz: false
        duck: true
```

Stage 3/8: Palindromic numbers

Description

In this stage, the program should check whether a number is a **Palindromic** one. A Palindromic number is symmetrical; in other words, it stays the same regardless of whether we read it from left or right. For example, 17371 is a palindromic number. 5 is also a palindromic number. 1234 is not. If read it from right, it becomes 4321. Add this new property to our program.

In previous stages, the program could process only one number. From now on, the program should accept a request from a user, analyze and execute it. The program should print `Enter a request` instead of asking for a natural number. The program should also continue execution unless a user enters a terminate command. Let's make it `0` (zero).

Your program should welcome users and print the instructions. At this point, make your program execute two commands. If a user enters a natural number, the program should indicate the properties of that number. If a user enters zero, then the program should exit. If a user enters a negative number by mistake, print an error message.

Objectives

In this stage, your program should:

1. Welcome users;
2. Display the instructions;
3. Ask for a request;
4. Terminate the program if a user enters zero;
5. If a number is not natural, print an error message;
6. Print the properties of the natural number;
7. Continue execution from step 3, after the request has been processed.

The properties are `even`, `odd`, `buzz`, `duck` and `palindromic`.
The tests won't check the order of properties, indentation, and spaces.
You may format numbers as you like. Please, add the information below:

Instructions:

```
Supported requests:
- enter a natural number to know its properties;
- enter 0 to exit.
```

Error message:

```
The first parameter should be a natural number or zero.
```

Examples

The greater-than symbol followed by a space (`> `) represents the user input. Note that it's not part of the input.

Example 1:

```
Welcome to Amazing Numbers!

Supported requests:
- enter a natural number to know its properties;
- enter 0 to exit.

Enter a request: > 9223372036854775807

Properties of 9,223,372,036,854,775,807
        even: false
         odd: true
        buzz: true
        duck: true
 palindromic: false

Enter a request: > 101

Properties of 101
        even: false
         odd: true
        buzz: false
        duck: true
 palindromic: true

Enter a request: > -56

The first parameter should be a natural number or zero.

Enter a request: > 0

Goodbye!
```

Stage 4/8: Gapful numbers

Description

In this stage, we are going to add one more property — **Gapful numbers**. It is a number that contains at least 3 digits and is divisible by the concatenation of its first and last digit **without a remainder**. 12 is not a Gapful number, as it has only two digits. 132 is a Gapful number, as **1**3**2** % 12 == 0. Another good example of a Gapful number is 7881, as **7**88**1 %** 71 == 0.

Until this stage, the program could process only one number at a time. Now, a user should be able to enter two numbers to obtain the properties of a list of numbers. Separate the numbers with one space. Space separates the first number in the list and the length of the list.
For, example. `100 2` tells the program to process two numbers: `100` and `101`. `1 100` means that the program is about to process 100 numbers, starting from `1`.
If a user enters one number, the program should work the same as in the previous stages. The program should analyze a number and print its properties. As before, if a user enters a single `0` (zero), terminate the program.
Information about each number should be printed in one line in the following format:

```
140 is even, buzz, duck, gapful
141 is odd, palindromic
```

So, the format is `{number} is {property}, {property}, ... {property}`

Objectives

Your program should process various user requests. In this stage, your program should:

1. Welcome users;
2. Display the instructions;
3. Ask for a request;
4. If a user enters zero, terminate the program;
5. If a user enters an empty request, print the instructions;
6. If numbers are not natural, print an error message;
7. If one number is entered, calculate and print the properties of this number;
8. For two numbers, print the list of numbers with properties;
9. Once the request is processed, continue execution from step 3.

In the current stage, the property names include `even`, `odd`, `buzz`, `duck`, `palindromic` and `gapful`.
The test won't check the order of properties, their indentation, and spaces. You may format numbers as you like. Please, add the information below:

Instructions

```
Supported requests:
- enter a natural number to know its properties;
- enter two natural numbers to obtain the properties of the list:
  * the first parameter represents a starting number;
  * the second parameter shows how many consecutive numbers are to be printed;
- separate the parameters with one space;
- enter 0 to exit.
```

Error messages

```
The first parameter should be a natural number or zero.
```

```
The second parameter should be a natural number.
```

Examples

The greater-than symbol followed by a space (`> `) represents the user input. Note that it's not part of the input.

Example 1:

```
Welcome to Amazing Numbers!

Supported requests:
- enter a natural number to know its properties;
- enter two natural numbers to obtain the properties of the list:
  * the first parameter represents a starting number;
  * the second parameter shows how many consecutive numbers are to be processed;
- separate the parameters with one space;
- enter 0 to exit.

Enter a request: > 7881

Properties of 7,881
        buzz: false
        duck: false
 palindromic: false
      gapful: true
        even: false
         odd: true

Enter a request: > 7880

Properties of 7,880
        buzz: false
        duck: true
 palindromic: false
      gapful: false
```

```
         even: true
          odd: false

Enter a request: > 105 5

              105 is buzz, duck, gapful, odd
              106 is duck, even
              107 is buzz, duck, odd
              108 is duck, gapful, even
              109 is duck, odd

Enter a request: > exit

The first parameter should be a natural number or zero.

Enter a request: >

Supported requests:
- enter a natural number to know its properties;
- enter two natural numbers to obtain the properties of the list:
  * the first parameter represents a starting number;
  * the second parameter shows how many consecutive numbers are to be processed;
- separate the parameters with one space;
- enter 0 to exit.

Enter a request: > 0

Goodbye!

Process finished with exit code 0
```

Stage 5/8: Spy numbers

Description

In this stage, we will add another property that is called a **Spy number**. A number is said to be Spy if the sum of all digits is equal to the product of all digits.

Our program calculates the properties of numbers, and also knows how to process a list of numbers. But what if we want to find numbers that have a certain property? For example, in case we want to find ten Buzz numbers starting from 1000. We need to add a new request feature for this. In addition, add a new property — Spy numbers. These numbers hide well, so beware. Your task is to modify the program so that it can search for these numbers.

Objectives

Your program should process the user requests. In this stage, your program should:

1. Welcome users;

2. Display the instructions;
3. Ask for a request;
4. If a user enters zero, terminate the program;
5. If numbers are not natural, print an error message;
6. If a user inputs an incorrect property, print the error message and the list of available properties;
7. For one number, print the properties of the number;
8. For two numbers, print the list of numbers with their properties;
9. For two numbers and a property, print the list of numbers and their properties filtered by the indicated property;
10. Once a request is processed, continue execution from step 3.

In the current stage, the property names include `even`, `odd`, `buzz`, `duck`, `palindromic`, `gapful`, and `spy`.

The test won't check the order of properties, their indentation, and spaces. You may format numbers as you like. Please, add the information below:

Instructions

```
Supported requests:
- enter a natural number to know its properties;
- enter two natural numbers to obtain the properties of the list:
  * the first parameter represents a starting number;
  * the second parameter shows how many consecutive numbers are to be printed;
- two natural numbers and a property to search for;
- separate the parameters with one space;
- enter 0 to exit.
```

Error messages

```
The first parameter should be a natural number or zero.
```

```
The second parameter should be a natural number.
```

```
The property [SUN] is wrong.
Available properties: [EVEN, ODD, BUZZ, DUCK, PALINDROMIC, GAPFUL, SPY]
```

Examples

The greater-than symbol followed by a space (`> `) represents the user input. Note that it's not part of the input.

Example 1:

```
Welcome to Amazing Numbers!

Supported requests:
- enter a natural number to know its properties;
- enter two natural numbers to obtain the properties of the list:
  * the first parameter represents a starting number;
  * the second parameters show how many consecutive numbers are to be processed;
- two natural numbers and a property to search for;
- separate the parameters with one space;
- enter 0 to exit.

Enter a request: 9

Properties of 9
        buzz: false
        duck: false
 palindromic: true
      gapful: false
         spy: true
        even: false
         odd: true

Enter a request: 9 7

               9 is palindromic, spy, odd
              10 is duck, even
              11 is palindromic, odd
              12 is even
              13 is odd
              14 is buzz, even
              15 is odd

Enter a request: 99 9 spy

             123 is spy, odd
             132 is gapful, spy, even
             213 is spy, odd
             231 is buzz, gapful, spy, odd
             312 is spy, even
             321 is spy, odd
           1,124 is spy, even
           1,142 is spy, even
           1,214 is spy, even

Enter a request: 9 3 drake

The property [DRAKE] is wrong.
Available properties: [BUZZ, DUCK, PALINDROMIC, GAPFUL, SPY, EVEN, ODD]

Enter a request: 9 3 duck

              10 is duck, even
              20 is duck, even
              30 is duck, even

Enter a request: bye
```

```
The first parameter should be a natural number or zero.

Enter a request: 0

Goodbye!

Process finished with exit code 0
```

Stage 6/8: Sunny and Square numbers

Description

N is a **sunny** number if N+1 is a **perfect square** number. In mathematics, a square number
or a perfect square is an integer that is the square of an integer; in other words, it is the product of an integer with itself. For example, 9 is a square number, since it equals 32 and can be written as 3×3.

Our program can search for Spy and Palindromic numbers. What if we want to find all even Spy numbers? Or find all odd numbers among Palindromic numbers? Are there any Palindromics that are also Spies? In this stage, you will add the ability to search for several properties at once!

What if a user enters the same property twice by mistake? For example, they want to find all even numbers that are even? Just ignore this duplicate property. What about two mutually exclusive properties? For example, if a user wants to find even numbers that are odd. In this case, the program will initiate the search, but there are no such numbers. We need to make our program foolproof. If a request contains mutually exclusive properties, the program should abort this request and warn the user. There are three pairs of mutually exclusive properties. A request cannot include **Even and Odd**, **Duck and Spy**, **Sunny and Square** numbers.

Objectives

Your program should process the user requests. In this stage, your program should:

1. Welcome users;
2. Display the instructions;
3. Ask for a request;

4. If a user enters zero, terminate the program;
5. If numbers are not natural, print the error message;
6. If an incorrect property is specified, print the error message and the list of available properties;
7. For one number, calculate and print the properties of the number;
8. For two numbers, print the list of numbers with their properties;
9. For two numbers and one property, print the numbers with this property only;
10. For two numbers and two properties, print the numbers that have both properties.
11. If a user specifies mutually exclusive properties, abort the request and warn a user.
12. Once a request has been processed, continue execution from step 3.

In the current stage, the property names include `even`, `odd`, `buzz`, `duck`, `palindromic`, `gapful`, `spy`, `square`, and `sunny`.

The test won't check the order of properties, their indentation, and spaces. You may format numbers as you like. Please, add the information below:

Instructions

```
Supported requests:
- enter a natural number to know its properties;
- enter two natural numbers to obtain the properties of the list:
  * the first parameter represents a starting number;
  * the second parameter shows how many consecutive numbers are to be printed;
- two natural numbers and a property to search for;
- two natural numbers and two properties to search for;
- separate the parameters with one space;
- enter 0 to exit.
```

Error messages

```
The first parameter should be a natural number or zero.
```

```
The second parameter should be a natural number.
```

```
The property [SUN] is wrong.
Available properties: [EVEN, ODD, BUZZ, DUCK, PALINDROMIC, GAPFUL, SPY, SQUARE, SUNNY]
```

```
The properties [HOT, SUN] are wrong.
Available properties: [EVEN, ODD, BUZZ, DUCK, PALINDROMIC, GAPFUL, SPY, SQUARE, SUNNY]
```

`The request contains mutually exclusive properties: [ODD, EVEN]`
`There are no numbers with these properties.`

Examples

The greater-than symbol followed by a space (`> `) represents the user input. Note that it's not part of the input.

Example 1:

```
Welcome to Amazing Numbers!

Supported requests:
- enter a natural number to know its properties;
- enter two natural numbers to obtain the properties of the list:
  * the first parameter represents a starting number;
  * the second parameters show how many consecutive numbers are to be processed;
- two natural numbers and two properties to search for;
- separate the parameters with one space;
- enter 0 to exit.

Enter a request: > 1

Properties of 1
        buzz: false
        duck: false
 palindromic: true
      gapful: false
         spy: true
      square: true
       sunny: false
        even: false
         odd: true

Enter a request: > 1 8 square

               1 is palindromic, spy, square, odd
               4 is palindromic, spy, square, even
               9 is palindromic, spy, square, odd
              16 is square, even
              25 is square, odd
              36 is square, even
              49 is buzz, square, odd
              64 is square, even

Enter a request: > 1 7 sunny

               3 is palindromic, spy, sunny, odd
               8 is palindromic, spy, sunny, even
              15 is sunny, odd
              24 is sunny, even
              35 is buzz, sunny, odd
              48 is sunny, even
              63 is buzz, sunny, odd
```

```
Enter a request: > 1 7 sunny square

The request contains mutually exclusive properties: [SQUARE, SUNNY]
There are no numbers with these properties.

Enter a request: > 1 2 sunny polindromic

The property [POLINDROMIC] is wrong.
Available properties: [BUZZ, DUCK, PALINDROMIC, GAPFUL, SPY, SQUARE, SUNNY, EVEN, ODD]

Enter a request: > 1 2 sunny palindromic

               3 is palindromic, spy, sunny, odd
               8 is palindromic, spy, sunny, even

Enter a request: 0

Goodbye!

Process finished with exit code 0
```