Nim

There are many variations on this classic game, but the secret is always finding the right way to count.

The rules:

- ***Start with a positive whole number (eg 10), and two players.***
- ***Players take it in turns to subtract either 1 or 2 from the total.***
- ***The player who reduces the value to 0 (ie, removes the last one) <u>loses</u>.***

This game can be played quite easily with physical objects:

- *Place 10 pencils on the desk between you.*
- *Take it in turns to pick up either 1 or 2 pencils from the pile in the middle.*
- *Whoever takes the last pencil loses.*

Try playing this game in pairs.
Vary the number you start with, and vary which player goes first.
Answer the following:

1. **Does it matter who starts?**

2. **Does it matter what number you start with?**

3. **Are there any numbers that you *don't* want to start with?**

4. **Is there any sure-fire strategy for winning the game?**

5. **What happens if you allow players to take 1, 2 or 3 each time?**

6. **What changes if the game is played with 3 players instead of 2?**

Turn over to see how to code it up …

Playing Nim *with Python*

The key strategy for this game, in its original form (2 players, subtract 1 or 2), is to always leave the other player with *one more than a multiple of 3*. If you can do this even once during the game, you can force the other player to always be left with a number of the form $3k + 1$, because if they take 1, you can take 2 and if they take 2, you can take 1. In due course, you'll be down to 7, then 4, then 1, and your opponent will have to take 1.

Interacting with the user

In previous programs, once you set the code running, it may complete the whole program without requiring any further instructions from the user, but in nearly all cases, user input is what makes a program useful and effective (and even if it isn't essential, allowing the user to define certain parameters without themselves editing the code is a big plus).

There are two aspects to getting user input: requesting input from the user, and processing it. Getting input can be as simple as printing a statement (eg a question) and waiting for a typed response:

The input function

```
1  n = input("What number do you want? ")
2
3  print(n)
4  print(type(n))
```

When **input** is called, it prints the text given, then waits for the user to enter a response via the keyboard. When they press Enter, the result (a string of characters) is returned as the result of the function. This is then assigned to a variable which we can work with.
*Note that printing **type(n)** lets us see that the value we have assigned to **n** is not an **int**, but a **str**.*

Parsing the input

```
1  n = int(input("What number do you want? "))
2
3  for i in range(n):
4      print(i)
```

If we want a number to work with, we can convert it as soon as it is entered. Wrap the **input** function in the **int** function, and (if possible) the string will be converted to an integer type.
*Note that if the user enters **6.8**, or **five**, the program will crash because it can't convert those directly to integers.*

Try / Except

```
1  try:
2      n = int(input("What number do you want? "))
3  except:
4      n = 10
5      print("Didn't catch that.  We'll use 10")
6
7  for i in range(n):
8      print(i)
```

Dealing with errors (or even malicious attempts to hack your program) is an important part of any proper code.
This code 'tries' to run the code inside the **try** block, but if it fails, whatever is specified in the **except** block is carried out instead.
*In this case, we just pick a value to use, but in some circumstances, you might want to enclose the **try / except** code block inside a loop which repeats until a valid input is received.*

Playing the game: processing player moves

```
1   print("Welcome to NIM, the game I shall WIN!")
2   try:
3     n = int(input("What number do you want to start from? "))
4   except:
5     n = 10
6     print("Didn't catch that.  We'll use 10 :)")
7
8   print(f"Starting with {n}, take 1 or 2.  Don't take the last!")
9
10  while n >= 1:
11    try:
12      player_choice = int(input("Take 1 or 2? "))
13    except:
14      player_choice = 1
15    if player_choice not in [1, 2] or player_choice > n:
16      player_choice = 1
17    n -= player_choice
18    print(f"You took {player_choice}, leaving {n}.")
19    if n <= 0:
20      print("You lose!")
21      break
22    if n % 3 == 1:
23      bot_choice = 1
24    else:
25      bot_choice = (n - 1) % 3
26    n -= bot_choice
27    print(f"...I took {bot_choice}, leaving {n}.")
28    if n <= 0:
29      print("You win!")
```

After getting input from the player about what number of objects to start with, the main **while** loop of the program begins. This loop will continue until the condition (**n >= 1**) is met, or until it reaches a **break** statement.

The **try / except** here ensures that the player's choice is only 1 or 2 *(what happens if they enter something else?)*

The number is updated, and information about the current 'game state' is printed to the screen.

Playing the game: choosing computer moves

```
1   print("Welcome to NIM, the game I shall WIN!")
2   try:
3     n = int(input("What number do you want to start from? "))
4   except:
5     n = 10
6     print("Didn't catch that.  We'll use 10 :)")
7
8   print(f"Starting with {n}, take 1 or 2.  Don't take the last!")
9
10  while n >= 1:
11    try:
12      player_choice = int(input("Take 1 or 2? "))
13    except:
14      player_choice = 1
15    if player_choice not in [1, 2] or player_choice > n:
16      player_choice = 1
17    n -= player_choice
18    print(f"You took {player_choice}, leaving {n}.")
19    if n <= 0:
20      print("You lose!")
21      break
22    if n % 3 == 1:
23      bot_choice = 1
24    else:
25      bot_choice = (n - 1) % 3
26    n -= bot_choice
27    print(f"...I took {bot_choice}, leaving {n}.")
28    if n <= 0:
29      print("You win!")
```

The next part of the code deals with the computer's choices. First, we check to see if the game is already done (maybe the player took the last one – if so, they lost, and the loop can finish immediately), and if not, we test to see if the number of moves is a 'bad number' (ie, $3k + 1$). If not, the bot's choice is the number that will leave the player on a bad number themselves. *(what happens if the bot is stuck on a bad number?)*

Finally, after the computer's move, information is printed to the screen, and we check once more to see if the game is done. If so, the player won (since the bot made the final move).

Adapting the code

The game played above has some customisation: we can choose what number we start from, for instance. But how about allowing more flexibility? What if we let the user choose the maximum number to take away? How would the game be different if we could subtract 1, 2, 3, 4 or 5 each time? Can we code a more general strategy?

Additional mini-projects

Now you know how to accept and parse (verify and appropriately process) user input, you could make other interactive programs.

Quadratic Solver: Build a program that solves quadratic equations. It should ask the user for the values of a, b and c, check to make sure they are all valid numbers (in this case, **float** should be acceptable, not just **int**), and return solutions, where possible. Make sure it doesn't crash if there are no solutions!

Guess My Number: Build a program that guesses the user's secret number through as few yes or no questions as possible. With care, you should be able to guess any 6-digit number in 20 guesses.

Factor Finder: Build a program that tells you about the user's chosen number. For instance, it could find all factor pairs, write out the prime factorisation of the number, or return the total number of different factors of the number. The *totient* of a number is the number of different values which are less than the original number and also coprime to it (for instance, the totient of 10 is 4, because only 1, 3, 7 and 9 share no common factors with 10).

Pythagorean Triple Generator: It is possible to construct a Pythagorean triple from any positive integer starting value. If the input value is n, then $n^2 + 1$, $n^2 - 1$ and $2n$ are three integers which form a Pythagorean triple. Let the user specify values, and have your program generate and display the relevant triple (ideally in order, and maybe including the squares of the numbers so they can be easily verified).