Example

Write a program which sums up all integers from 1 to 100.

• In math, the question can be written as:

\[ \text{sum} = 1 + 2 + \cdots + 100. \]

• But this form is not doable in the machine.\(^1\)

\(^1\)We need to develop computational thinking. Read
http://rsta.royalsocietypublishing.org/content/366/1881/3717.full
or
http://blog.orangeapple.tw/posts/what-is-computational-thinking/.
• Normally, the machine executes the instructions **sequentially**.
• So one needs to decompose the math equation into several steps, like:

```java
...  
   int sum = 0;
   sum = sum + 1;
   sum = sum + 2;
   .
   .
   .
   sum = sum + 100;
...
```

• It is obvious that many similar statements can be found.
• Using a *while* loop, the program can be rearranged as follows:

```java
...  
 int sum = 0;
 int i = 1;
 while (i <= 100) {
     sum = sum + i;
     ++i;
 }
...
```

• You should guarantee that the loop will terminate as expected.

• In practice, the number of loop steps (iterations) is *unknown* until the input data is given.
Malfunctioned Loops

- It is really easy to make an infinite loop.

```
... while (true);
...```

- The common errors of the loops are:
  - never start
  - never stop
  - not complete
  - exceed the expected number of iterations
Example

Write a program which asks the sum of two random integers and lets the user repeatedly enter a new answer until correct.

```java
Scanner input = new Scanner(System.in);
int x = (int) (Math.random() * 10);
int y = (int) (Math.random() * 10);
int ans = x + y;

System.out.println(x + " + " + y + " = ? ");
int z = input.nextInt();

while (z != ans) {
    System.out.println("Try again? ");
    z = input.nextInt();
}
System.out.println("Correct.");
input.close();
```
Loop Design Strategy

- Writing a correct loop is not an easy task for novice programmers.
- Consider 3 steps when writing a loop:
  - **Find the pattern:** identify the statements that need to be repeated.
  - **Wrap by loops:** put these statements in the loop.
  - **Set the continuation condition:** translate the criterion from the real world problem into computational conditions.$^2$

$^2$Not unique.
Another common technique for controlling a loop is to designate a special value when reading and processing a set of values.

- This special input value, known as a *sentinel value*, signifies the end of the loop.
- For example, the operating systems and the GUI apps.
Example: Cashier Problem

Write a program which sums over positive integers from consecutive inputs and then outputs the sum when the input is nonpositive.

```java
...  
    int sum = 0;
    Scanner input = new Scanner(System.in);
    System.out.println("Enter price?");
    int price = input.nextInt();
    while (price > 0) {
        sum += price;
        System.out.println("Enter price?");
        price = input.nextInt();
    }
    System.out.println("Total = " + sum);
    input.close();
...  
```

- Line 8 and 9 are the recurrence of Line 3 and 4?!
do-while Loops

A do-while loop is similar to a while loop except that it does execute the loop body first and then checks the loop continuation condition.

```java
... do {
    // loop body
} while (condition); // Do not miss the semicolon!
...```

- Note that there is a semicolon at the end of the do-while loop.
- The do-while loops are also called posttest loops, in contrast to while loops, which are pretest loops.
Flowchart:

- Start
- Statement(s) (loop body)
- Loop-continuation-condition?
  - true
    - Statement(s) (loop body)
  - false
    - End
Write a program which sums over positive integers from consecutive inputs and then outputs the sum when the input is nonpositive.

```java
... int sum = 0, price = 0;
Scanner input = new Scanner(System.in);
   do {
      sum += price;
      System.out.println("Enter price?");
      price = input.nextInt();
   } while (price > 0);
System.out.println("Total = " + sum);
input.close();
...
A for loop generally uses a variable to control how many times the loop body is executed.

```java
... for (init_action; condition; increment) {
    // loop body
}
...```

- **init-action**: declare and initialize a variable
- **condition**: set a criterion for loop continuation
- **increment**: how the variable changes after each iteration
- Note that these terms are separated by semicolons.
Example

Sum from 1 to 100

Write a program which sums from 1 to 100.

```java
... 
  int sum = 0;
  for (int i = 1; i <= 100; ++i)
    sum = sum + i;
... 
```

- Compared to the while version,

```java
... 
  int sum = 0;
  int i = 1;
  while (i <= 100) {
    sum = sum + i;
    ++i;
  }
... 
```
Example: Selection Resided in Loop

Display all even numbers

Write a program which displays all even numbers smaller than 100.

- An even number is an integer of the form $x = 2k$, where $k$ is an integer.
• You may use the modular operator (%).

```java
... for (int i = 1; i <= 100; i++) {
    if (i % 2 == 0) System.out.println(i);
} ...
```

• Also consider this alternative:

```java
... for (int i = 2; i <= 100; i += 2) {
    System.out.println(i);
} ...
```

• How about odd numbers?
Example: Monte Carlo Simulation

- Write a program which conducts a Monte Carlo simulation to estimate $\pi$.

See https://en.wikipedia.org/wiki/Monte_Carlo_method.
Jump Statements

The keyword `break` and `continue` are often used in repetition structures to provide additional controls.

- **break**: the loop is terminated right after a `break` statement is executed.
- **continue**: the loop skips this iteration right after a `continue` statement is executed.
- In practice, jump statements in loops should be conditioned.
Example: Primality

Write a program which determines if the input integer is a prime number.

• Let \( x > 1 \) be any natural number.
• Then \( x \) is said to be a prime number if \( x \) has no positive divisors other than 1 and itself.
• It is then straightforward to check if it is prime by dividing \( x \) by all natural numbers smaller than \( x \).
• For speedup, you can divide \( x \) by only numbers smaller than \( \sqrt{x} \). (Why?)
Scanner input = new Scanner(System.in);
System.out.println("Enter x > 2?");
int x = input.nextInt();
boolean isPrime = true;
input.close();

double upperBd = Math.sqrt(x);
for (int y = 2; y < upperBd; y++) {
    if (x % y == 0) {
        isPrime = false;
        break;
    }
}

if (isPrime) {
    System.out.println("Prime");
} else {
    System.out.println("Composite");
}