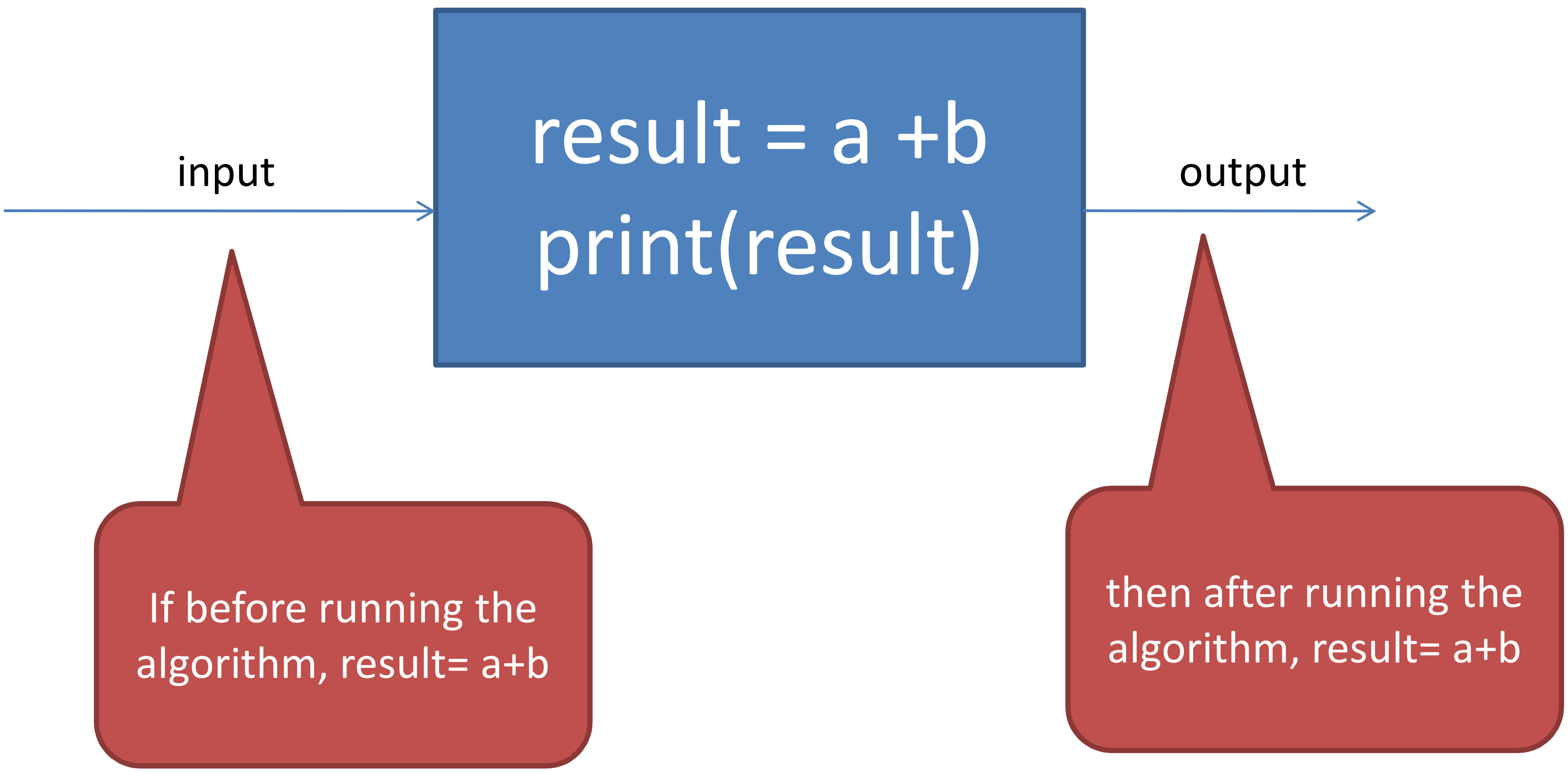
Ch2: Loop Invariants

305233, 305234

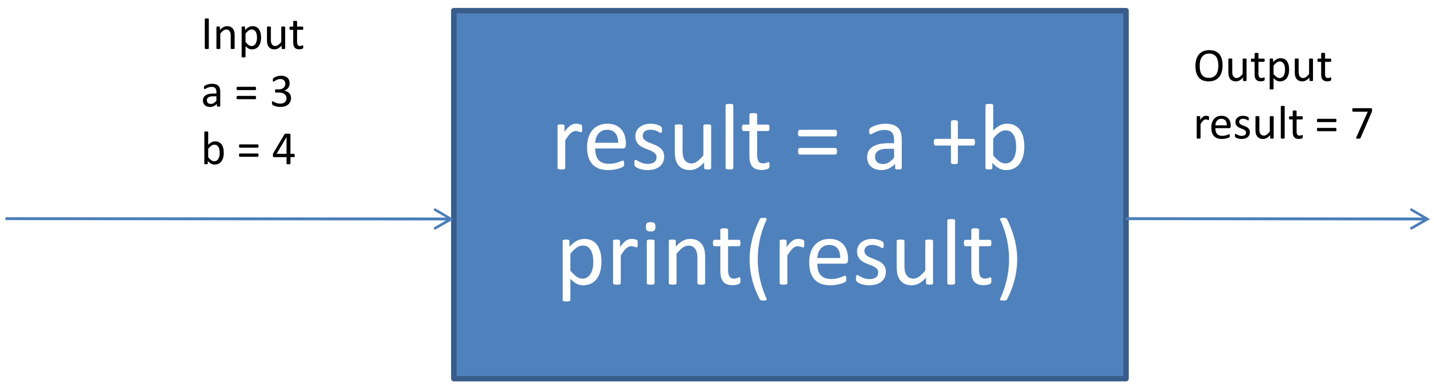
Algorithm Analysis and Design

Jiraporn Pooksook
Naresuan University

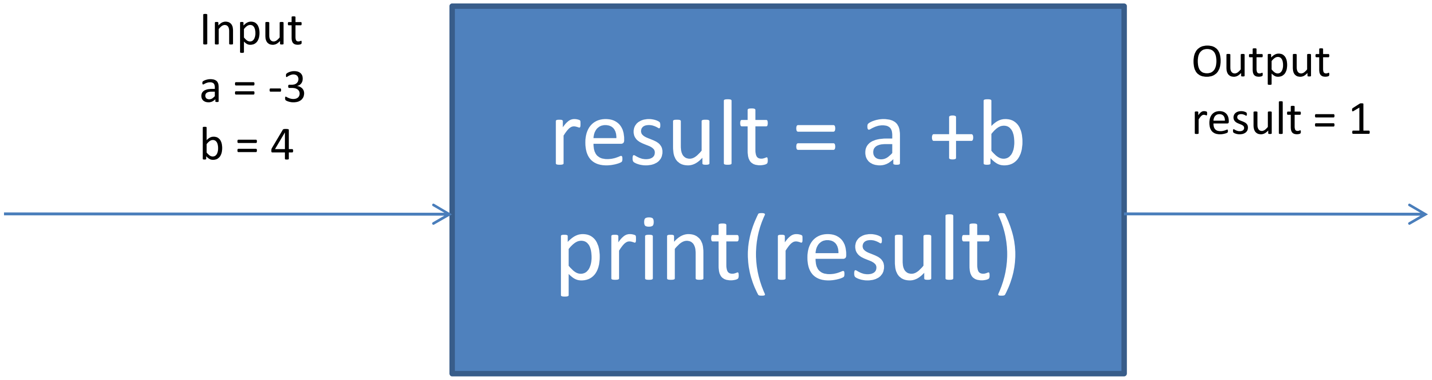
Accuracy of an Algorithm



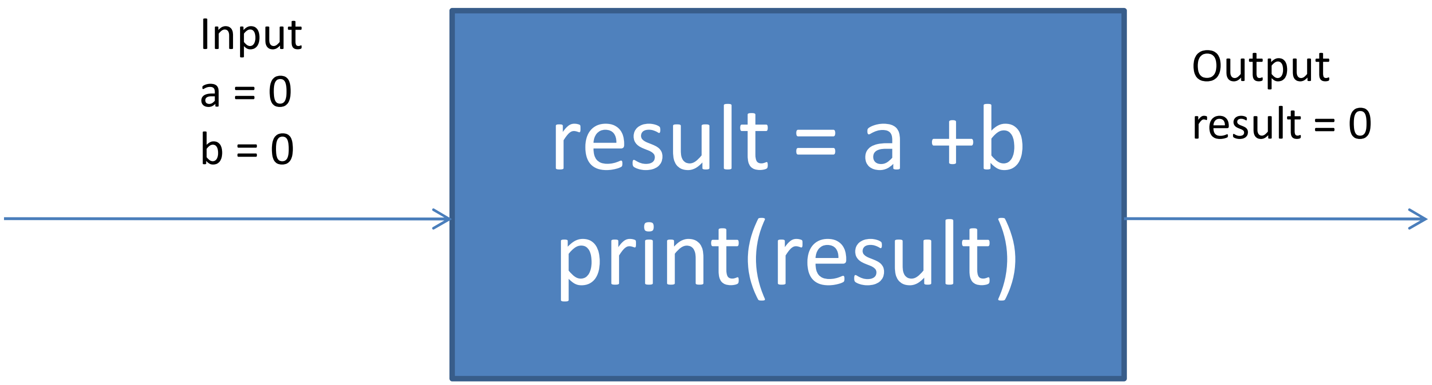
Accuracy of an Algorithm



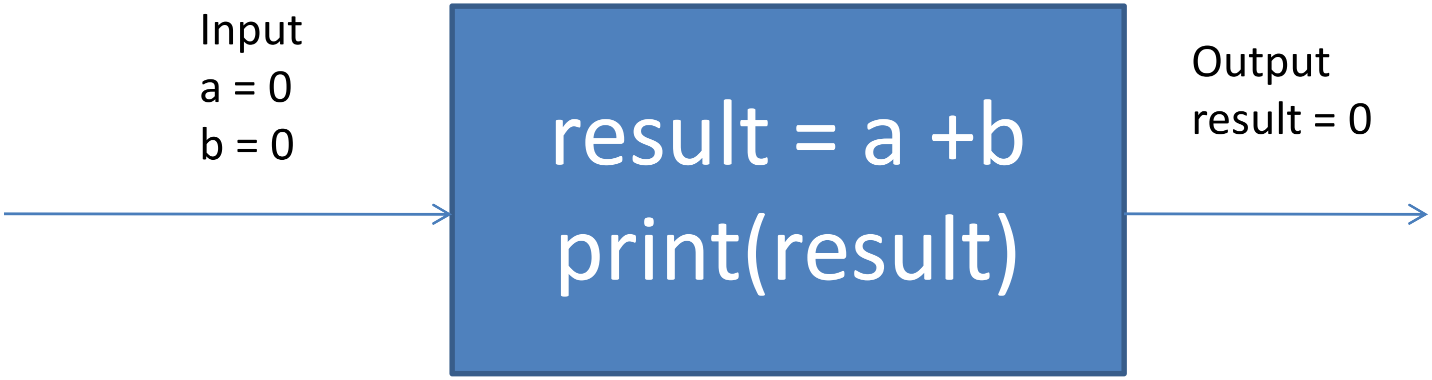
Accuracy of an Algorithm



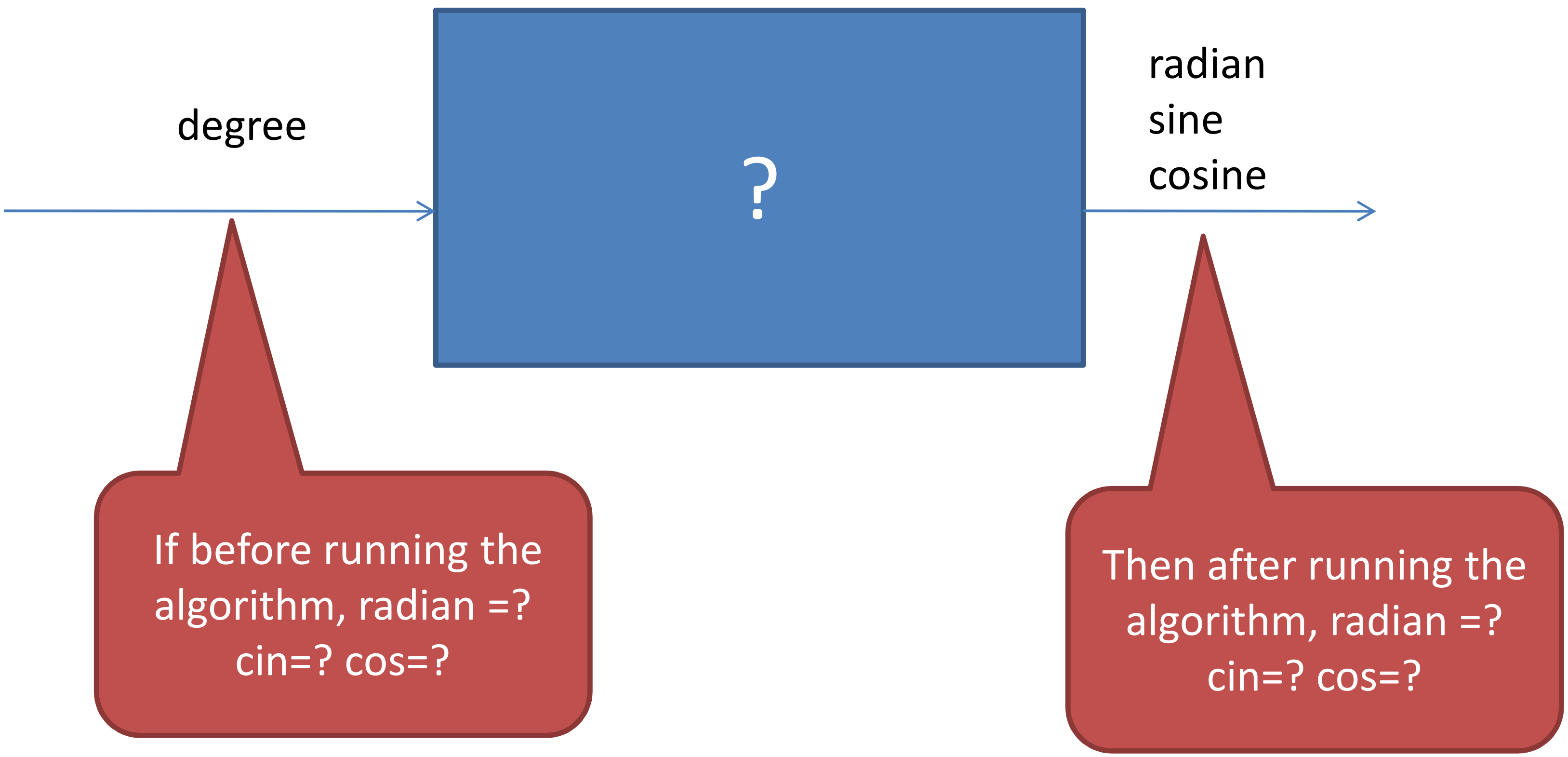
Accuracy of an Algorithm



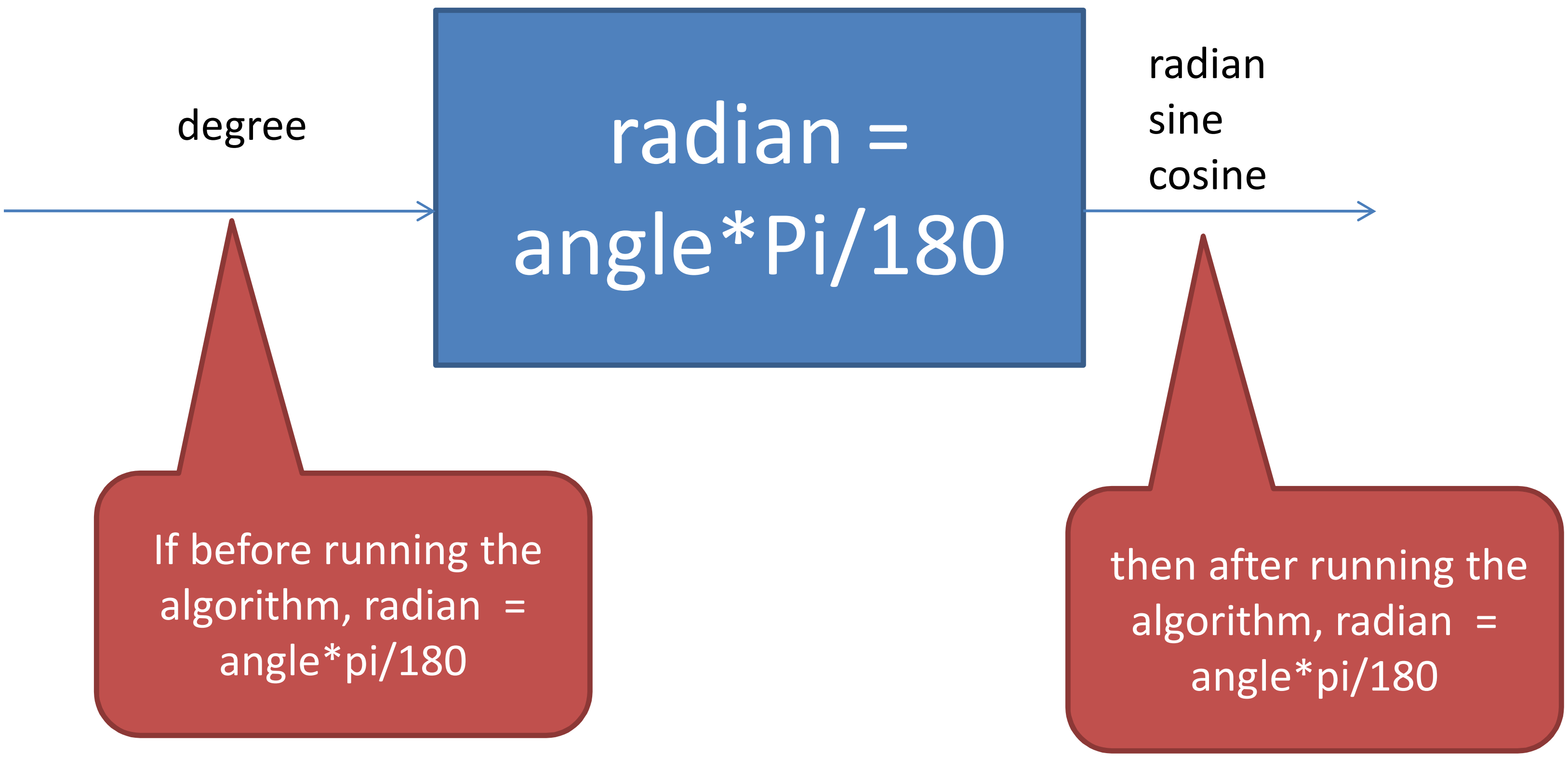
Accuracy of an Algorithm



Example: Accuracy of an Algorithm



Example: Accuracy of an Algorithm



Example

```
import math
a = float(input("Enter an angle in degrees: "))
r = a*(22/7)/180
print("%f degrees = %.2f radians and sin(%f) = %.2f and cos(%f) = %.2f" %
      (a,math.pi(a),math.sin(r),math.cos(r)))
```

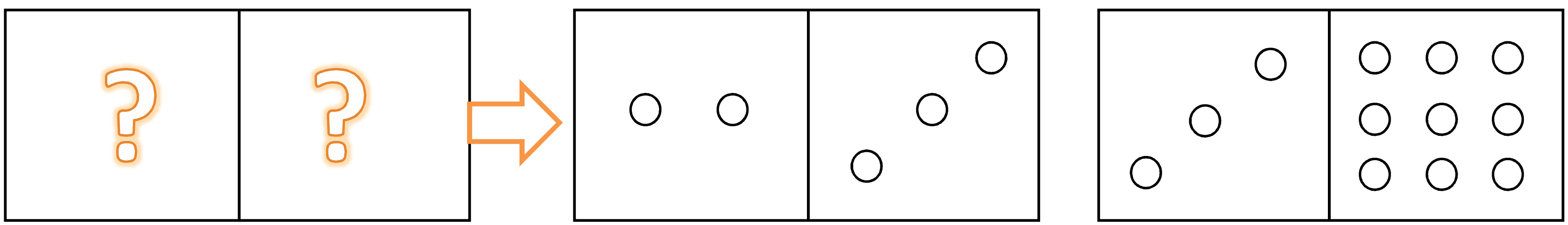
```
degree = int(input("Enter an angle in degrees: "))
import math
radian = (degree*math.pi)/180
sin = math.sin(radian)
cosine = math.cos(radian)
print("%d degrees = %.2f radians and sin(%d) = %.2f and cos(%d) = %.2f"%
      (degree,radian,degree,sin,degree,cosine))
```

Example

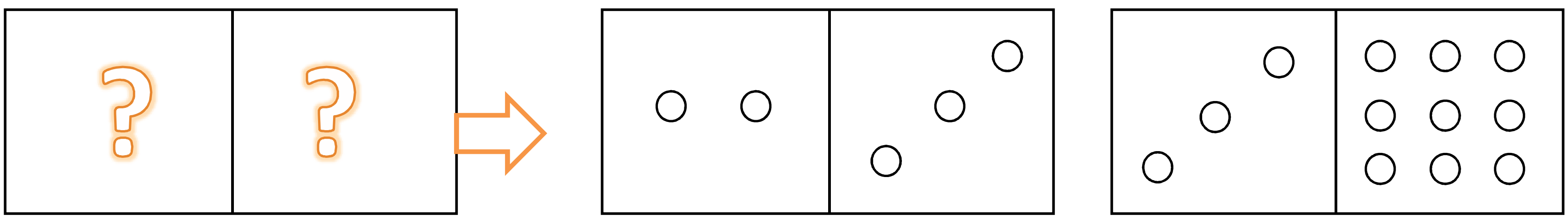
```
import math
pi = 3.14
angle = int(input("Enter an angle in degrees: "))
radian = angle*(pi)/180
print("%d degrees = %.2f radian" % angle, radian)
```

```
number = math.sin(input('Enter an angle in degrees:'))
Ra = (number*(math.pi))/180
math.sin = number
```

Example: Accuracy of an Algorithm



Example: Accuracy of an Algorithm



Domino tile
(left, right)

left/right =2
or
left/right = 9

Place Left / Right/ None

If before running the algorithm, right of input = left of given or left of input = right of given

then after running the algorithm, right of input = left of given or left of input = right of given

Example

```
print("Enter your domino tile: ")
x = int(input())
y = int(input())

if x==2 and y==9 or x==9 and y==2:
        print("Place your block the righ.")
        print("Place your block the left.")
elif x==9 or y==9:
       print("Place your block the righ.")
elif x==2 or y==2:
       print("Place your block the left.")
else:
       print("Place your block the tile.")
```

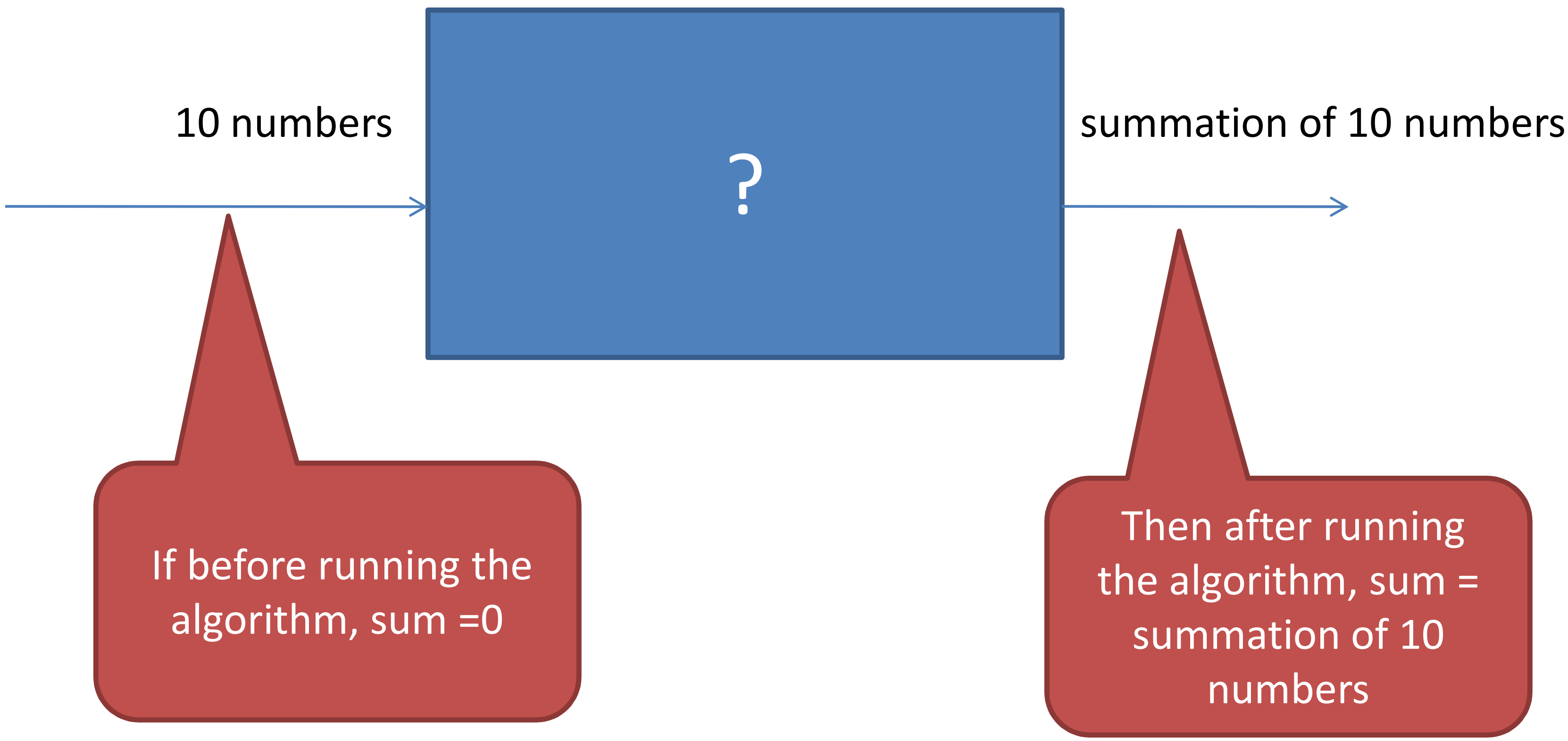
Example

```
x = int(input("Enter your domino title: = " ))
y = int(input("Enter your domino title: = " ))
if x==2 and y==9:
    print("place your block on the left amd right")
elif x==2 and y != 9:
         print("place your block on the left ")
elif x!=2 and y == 2:
         print("place your block on the left ")
elif x==2 and y ==2:
         print("place your block on the right ")
elif x==9 and y == 9:
         print("place your block on the left ")
elif x==9 and y == 2:
         print("place your block on the left amd right")
elif x!=2 and y == 9:
         print("place your block on the right ")
elif x!=9 and y == 9:
         print("place your block on the right ")
elif x!=9 and y == 2:
         print("place your block on the left ")
elif x!=9 and y != 2:
         print("cannot place your block tite  ")
elif x!=9 and y != 9:
         print("cannot place your block tite ")
elif x!=2 and y != 2:
         print("cannot place your block tite  ")
elif x!=2 and y != 9:
         print("cannot place your block tite ")
```

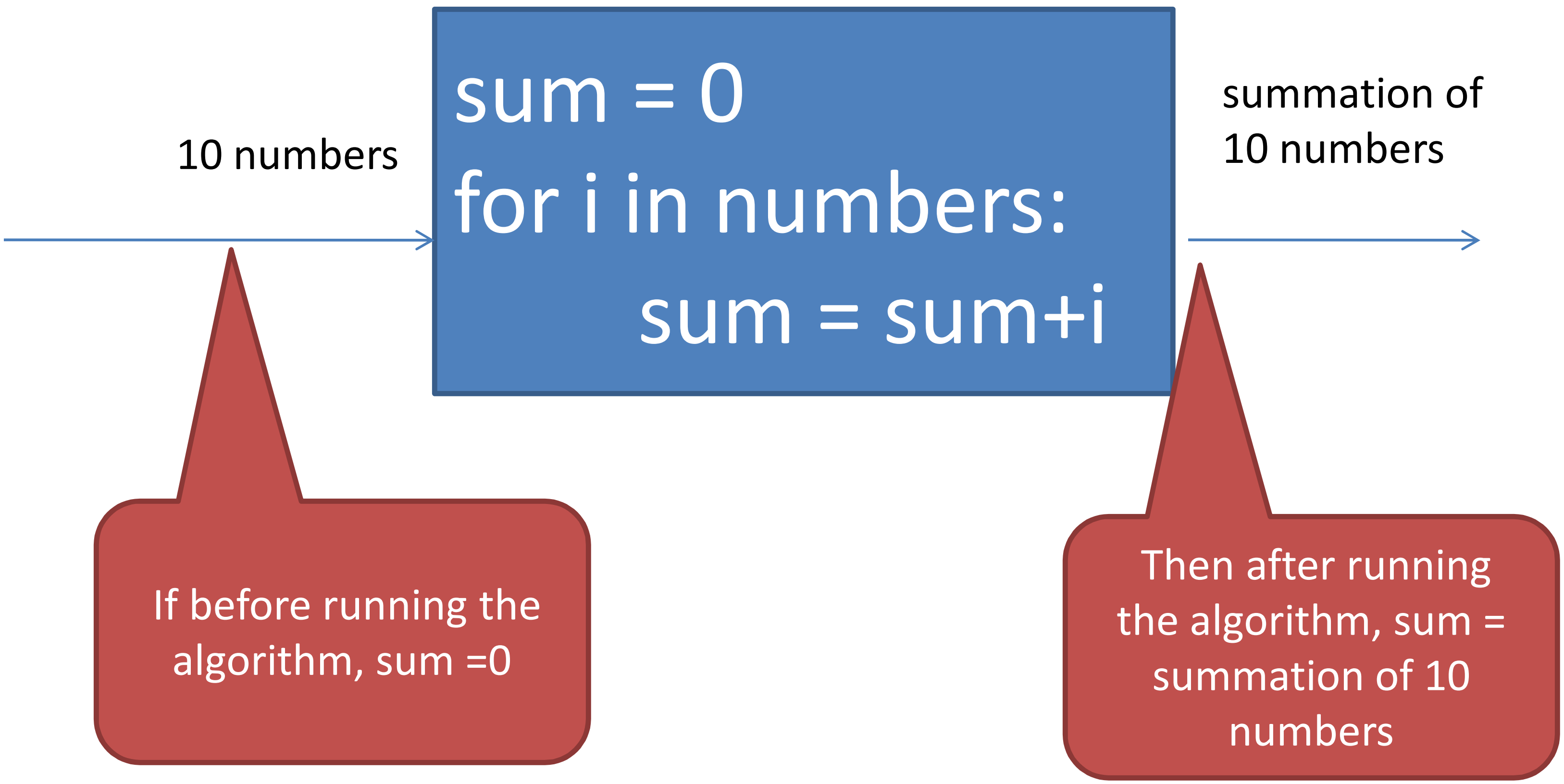
Example

```
print(int(input("Enter your domino tile : ")))
left = (int(input(" ")))
right = (int(input(" ")))

if left == 2 or right == 9:
    print("Place your block on the left.")
    print("Place your block on the right.")

elif right == 9:
    print("Place your block on the right.")

elif left == 2:
    print("Place your block on the left.")

else:
    print("Cannot place your domino tile.")
```

Example: Accuracy of an Algorithm



Example: Accuracy of an Algorithm



Example: Accuracy of an Algorithm

Round 1

[1,2,3,4,5,6,7,8,9,10]

```
sum = 0
for i in numbers:
        sum = sum+i
```

summation of
0 +1

If before running loop 1, sum =0

Then after running loop 1, sum = 1

Example: Accuracy of an Algorithm

Round 2

[1,2,3,4,5,6,7,8,9,10]

```
sum = 0
for i in numbers:
        sum = sum+i
```

summation of
0 +1 + 2

If before running loop 2, sum =1

Then after running loop 2, sum = 3

Example: Accuracy of an Algorithm

Round 3

[1,2,3,4,5,6,7,8,9,10]

```
sum = 0
for i in numbers:
        sum = sum+i
```

summation of
0 +1 + 2+3

If before running loop 3, sum =3

Then after running loop 3, sum = 6

Example: Accuracy of an Algorithm

Round 10

[1,2,3,4,5,6,7,8,9,10]

```
sum = 0
for i in numbers:
        sum = sum+i
```

summation of
0 +1 + 2+3+4+5
+6+7+8+9+10

If before running loop 10, sum =45

Then after running loop 10, sum = 55

Before running loop N, we have sum value of loop N-1

What is a Loop Invariant?

- An **loop invariant** is a formal statement of a properties of variables in an algorithm which holds true just before and after each iteration of running the **loop**.
- Similar to mathematical induction where the initialization is proving a base case and the maintenance is proving an inductive step.

proofs of a Loop Invariant

- **Initialization**
  - It is true prior to the first iteration of the loop.
- **Maintenance**
  - If it Is true before an iteration of the loop, it remains true before the next iteration.
- **Termination**
  - When the loop terminates, the invariant gives a useful property that helps shows that the algorithm is correct.

Example loop invariants with summation

```
sum = 0
for  i=1 to length[A]
        sum = sum + A[i]
```

What is a loop invariant for this code?

A property that will be true before and after running the loop.

A loop invariant is

before running loop i , $sum = \sum_{m=1}^{i-1} A[m]$

Example loop invariants with summation

Let us check with some sample input

Input = [9,5,7,4,2]

Initialization:
At i = 1 , m=1-1 = 0
hence, sum = 0
holds True!!!

```
sum = 0
for  i=1 to length[A]
        sum = sum + A[i]
```

Example loop invariants with summation

Maintenance:
If sum (before) = sum from 1 to i-1 then
sum(before next iter) = sum from 1 to i-1 +1

Input = [9,5,7,4,2]

```
sum = 0
for  i=1 to length[A]
        sum = sum + A[i]
```

i	Sum(before) sum to i - 1	Sum(after)
1	0	0 + 9
2	9	0+9+5
3	14	0+9+5+7
4	21	0+9+5+7+4
5	25	0+9+5+7+4+2
6	27	stop

Termination: sum from 1 to n
sum = 0+9+5+7+4+2
Holds True!!

Example loop invariants with summation

Let us check theoretically

```
sum = 0
for  i=1 to length[A]
        sum = sum + A[i]
```

A loop invariant is

before running at loop i , $sum = \sum_{m=1}^{i-1} A[m]$

Initialization: at loop 1, sum = 0 (True!!)

Maintenance:

If at before running loop i , sum = A[1]+A[2]+…+A[i-1]

then after running loop i , sum = A[1]+A[2]+…+A[i-1]+A[i]

Hence, before running loop i+1 , sum = A[1]+A[2]+…+A[i-1]+A[i] (True!!)

Termination:

Goal(output of program) => $sum = \sum_{i=1}^{n} A[i]$

At start of running at loop n+1, sum = A[1]+A[2]+…+A[n-1]+A[n] (True!!)

Exercise: Loop variant with Max Array

- Write a pseudo code of an algorithm for finding a maximal number in an array of size n.
- Write a proof of the correctness of the algorithm using loop invariants.

```
max = A[1]
for  i=2 to length[A]
        if max < A[i]
                max = A[i]
```

Solution: Loop variant with Max Array

```
max = A[1]
for  i=2 to length[A]
        if max < A[i]
                max = A[i]
```

Loop Invariant = Before running loop i , max is the largest number from A[1] to A[i-1]

Initialization:
Before running first loop where i=2 , max = A[1] which is the maximum number of A[2-1] (True!!)

Maintenance:
If before running loop i , max is the largest number among A[1] to A[i-1]
then   after running loop i, if max < A[i] then max = A[i] which is the largest of A[1… i]
if max > A[i] then max does not change and it is the largest of A[1…i].
Hence before running loop i+1, max is is the largest number among A[1] to A[i] (True!!)

Termination: at starting of loop n+1, max is the largest number among A[1] to A[n] (True!!)

Exercise: Insertion-Sort

A loop invariant =
all elements in A[1 … j – 1] are in sorted order.

initialization

```
for j=2 to length[A]
    do key = A[ j ]

    i = j - 1
    while i > 0 and A[ i ] > key
        do A[i+1] = A[ i ]
        i = i - 1
    A[i+1]=key
```

mainte-
nance

termination

Exercise loop invariants with insertion-sort

```
for j=2 to length[A]
        do key = A[ j ]

        i = j - 1
        while i > 0 and A[ i ] > key
                do A[i+1] = A[ i ]
                i = i - 1
        A[i+1]=key
```

Exercise loop invariants with insertion-sort

A loop invariant =
all elements in A[1 … j – 1] are in sorted order.

Input = [9,5,7,4,2]

j	key	A[1 to j-1] (before)	i	A[i] > key	A[1 to j-1] (after)	A[1… n]
2						
3						
4						
5						