CS111

Introduction to Computing Science

Previously

- Standard input and output
- Numeric types
- Decisions
- Multiple and nested decisions

This and next week

Loops

- To implement **`while`, `for` and `do`** loops
- To avoid infinite loops and off-by-one errors
- To understand nested loops

What Is the Purpose of a Loop?

A loop is a statement that is used to:

execute one or more statements
repeatedly until a goal is reached.

Sometimes these one-or-more statements
will not be executed at all
—if that's the way to reach the goal

The Three Loops in C++

C++ has these three looping statements:

`while`

`for`

`do`

The `while` Loop

In a stadium you walk in a loop as long as the music plays.

The `while` Loop

In a stadium you walk in a loop as long as the music plays.

The `while` Loop

In a stadium you walk in a loop as long as the music plays.

The `while` Loop

In a stadium you walk in a loop as long as the music plays.

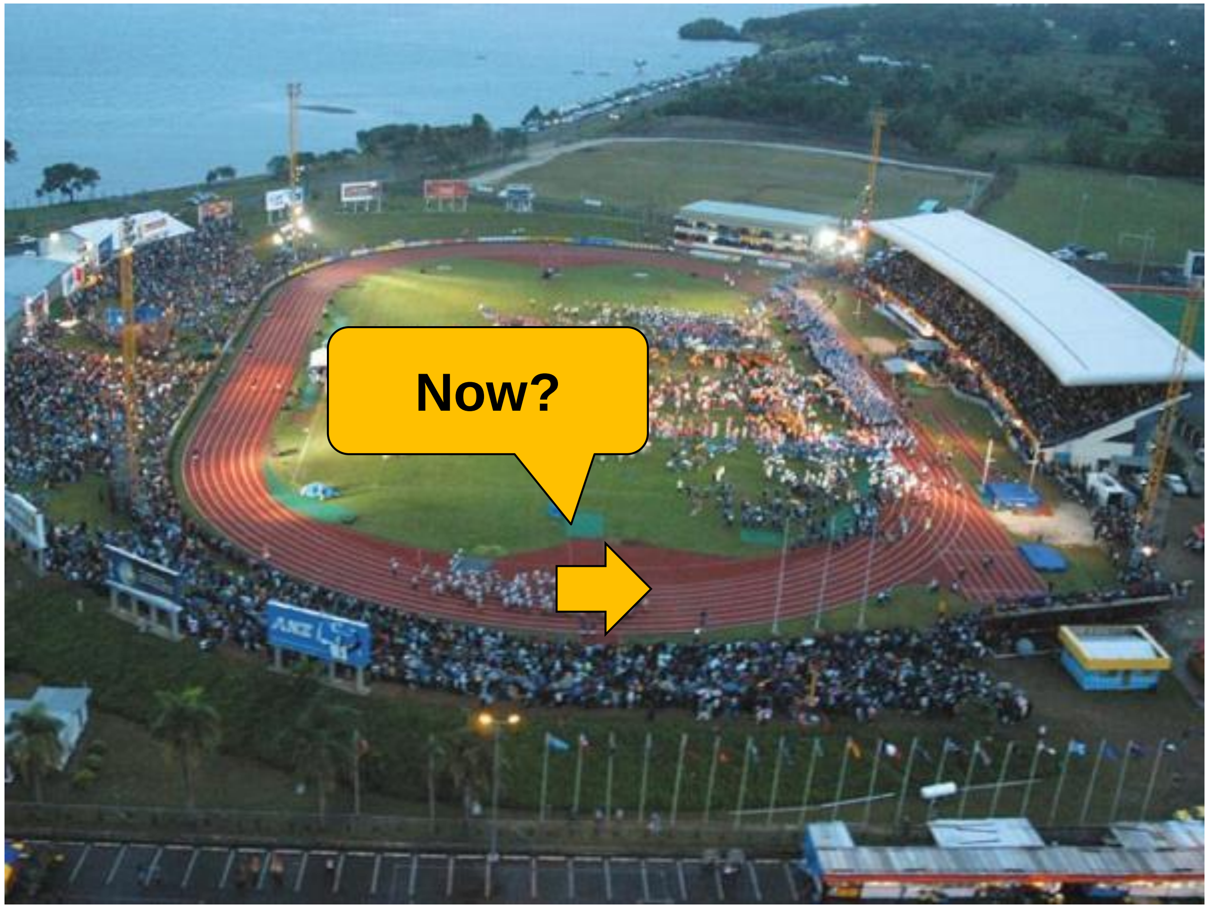
The `while` Loop

The **`while`** statement executes statements as long as a condition is true.

The `while` Loop

The **`while`** statement executes statements as long as a condition is true.

The `while` Loop



The **`while`** statement executes statements as long as a condition is true.

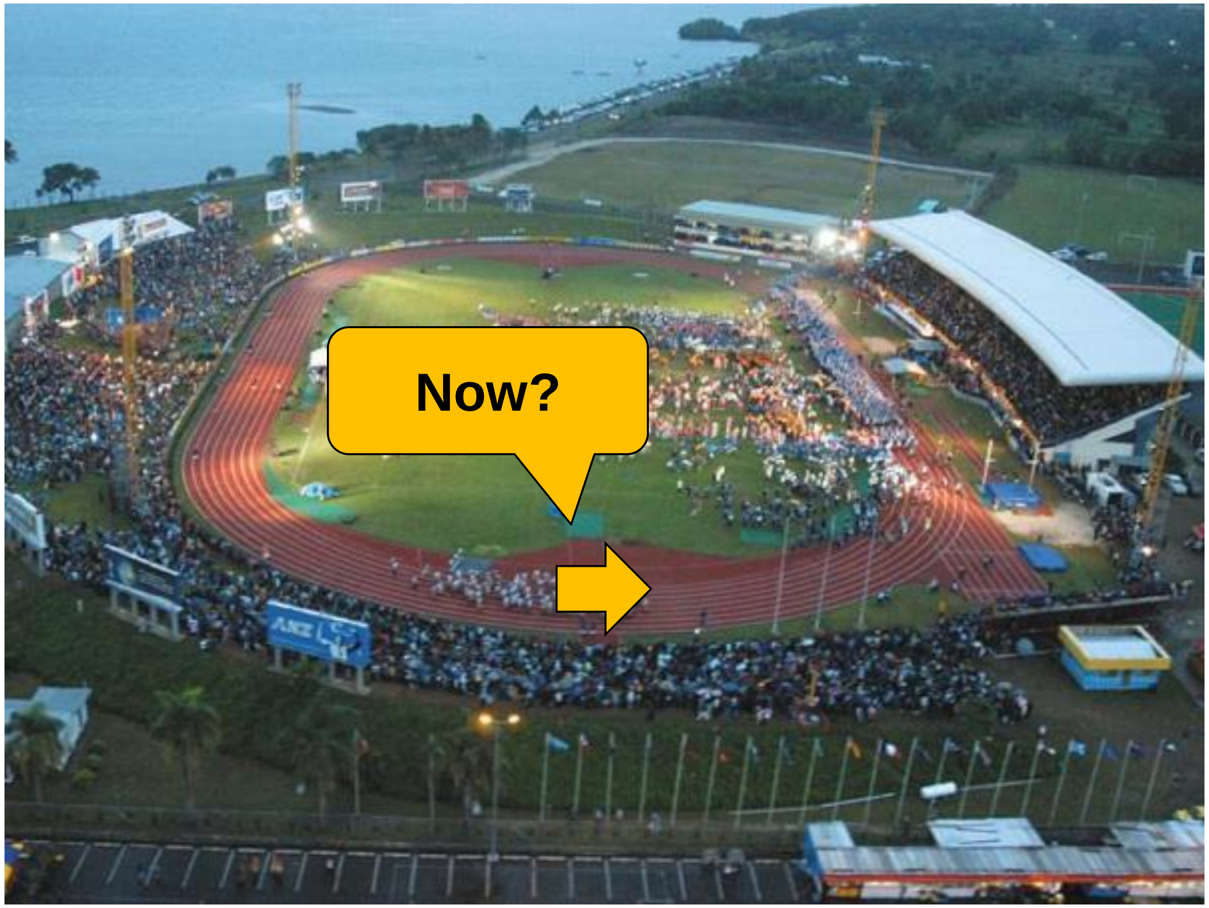
The **`while`** Loop

The **`while`** statement executes statements as long as a condition is true.

The `while` Loop

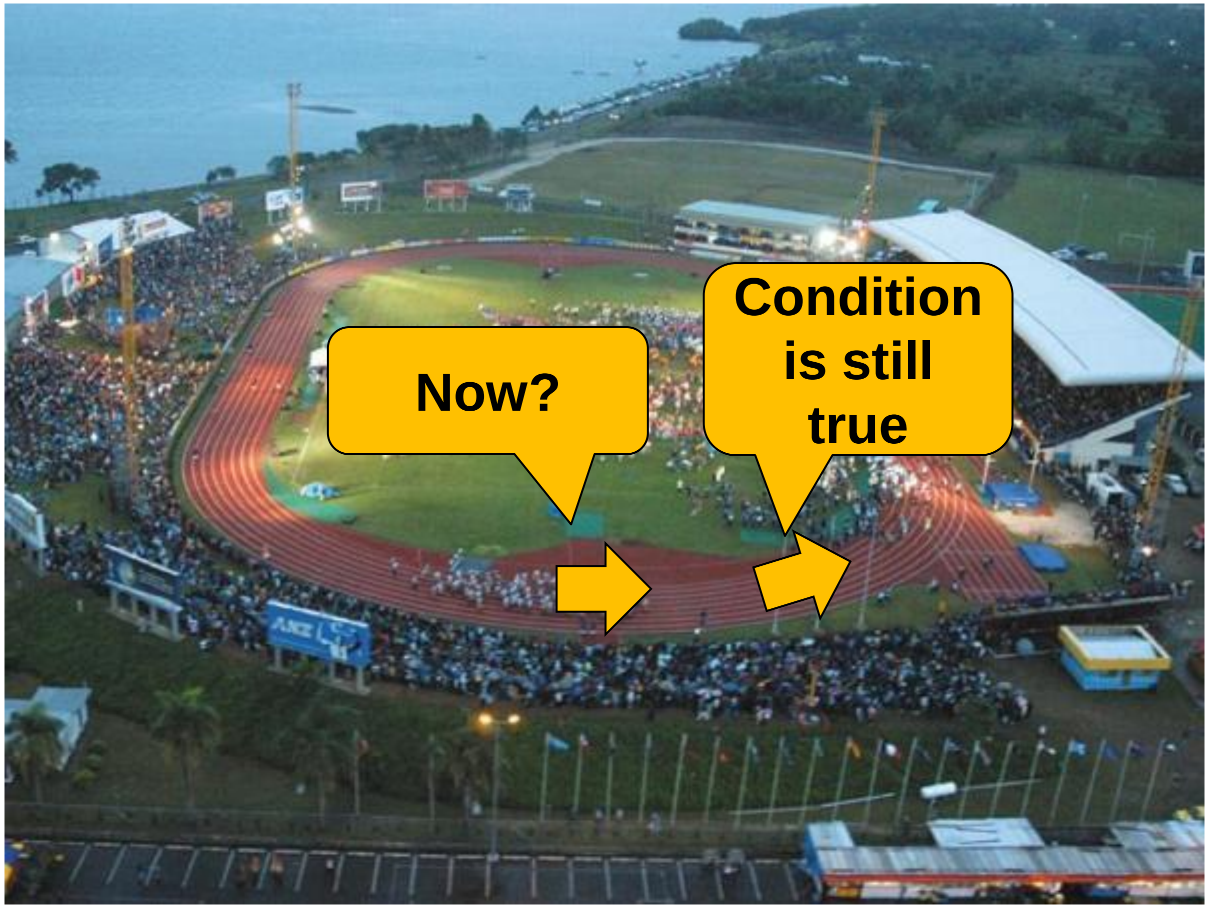
The **`while`** statement executes statements as long as a condition is true.

The `while` Loop



The **`while`** statement executes statements as long as a condition is true.

The `while` Loop



The **`while`** statement executes statements as long as a condition is true.

The `while` Loop

The **`while`** statement executes statements as long as a condition is true.

The `while` Loop

The **`while`** statement executes statements as long as a condition is true.

The `while` Loop

The **`while`** statement executes statements as long as a condition is true.

The `while` Loop

The **`while`** statement executes statements as long as a condition is true.

The `while` Loop

A while loop has this structure

```
while (condition)
{
   statements
}
```

- It starts with the keyword “while”
- Followed by a condition
- Followed by one or more statements.

The *condition* is some kind of test (the same as in the **if** statement)

The statements are repeatedly executed while the condition is true.

The statements are also called the **body** of the while.

The loop stops when the condition is false.

Using `while` Loop

An investment problem:

- Starting with $10,000, how many years until we have at least $20,000, if we get 5% interest.
- The algorithm:

1. Start with a year value of 0 and a balance of $10,000.
2. **Repeat** the following steps

   while the balance is less than $20,000:
   - Add 1 to the year value.
   - Compute the interest over the current balance.
   - Add the interest to the balance.
3. Report the final year value as the answer.

Using `while` Loop

A closer look at step 2

2. **<u>Repeat</u>** the following steps

   <u>while the balance is less than $20,000</u><u>:</u>

   - Add 1 to the year value.
   - Compute the interest over the current balance.
   - Add the interest to the balance.

- To get the answer adding and multiplying must be repeated some unknown number of times.
- While a condition is true.

Using `while` Loop

A closer look at the condition

2. **Repeat** the following steps

   while the balance is less than $20,000:

This can be implemented as:

```
while (balance < TARGET)
```

Using `while` Loop

A closer look at the body

- Add 1 to the year value.
- Compute the interest over the current balance.
- Add the interest to the balance.

This can be implemented as:

```
year++;
double interest = balance * RATE / 100;
balance = balance + interest;
```

Using `while` Loop

Put together we get

```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

- Note, that variable **`interest`** is defined **inside** the loop.
- Each iteration (round) will define a fresh variable **`interest`**.
- This means it will not be used outside.

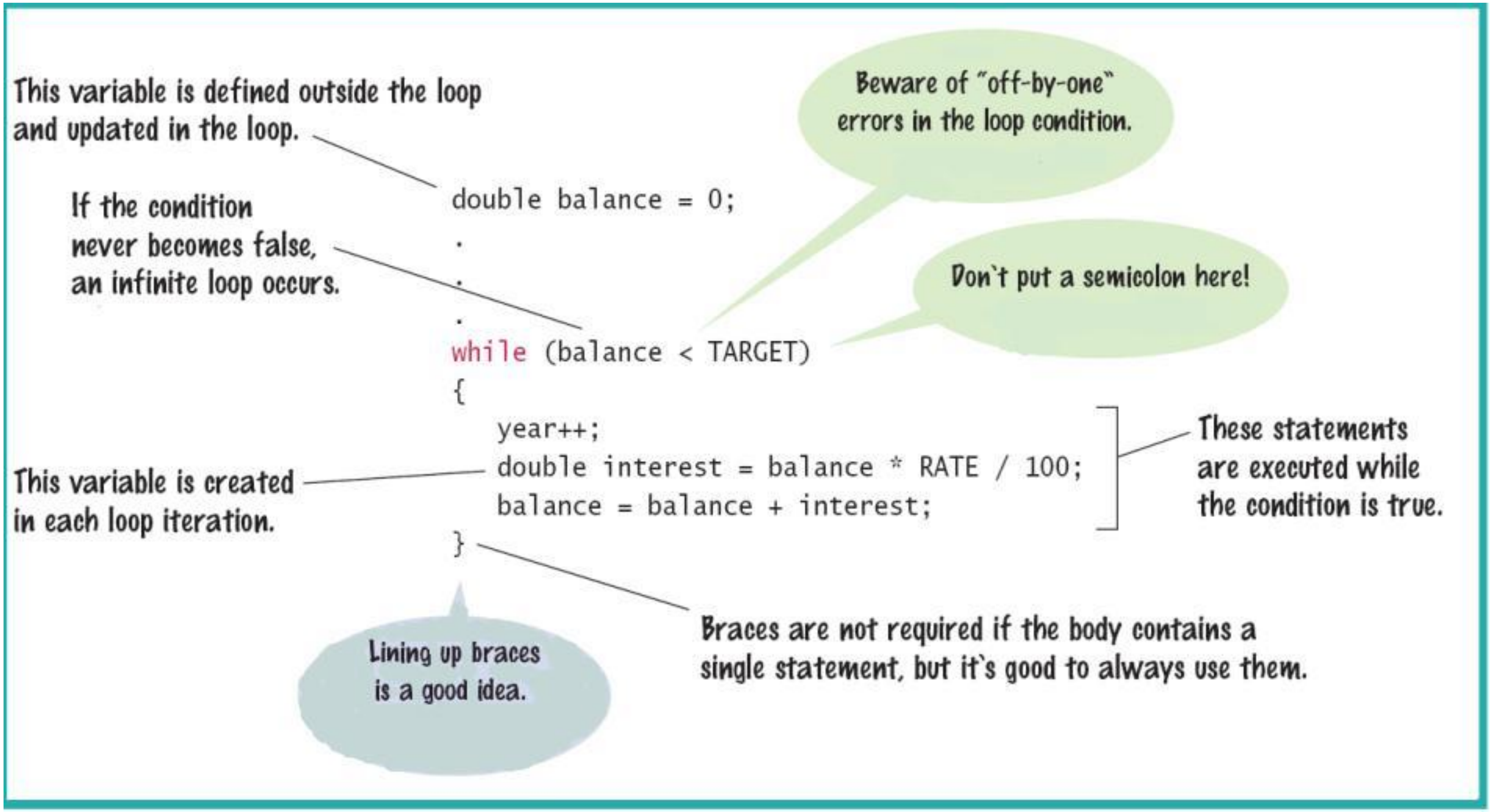
Using `while` Loop

Put together we get

```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

- Variable `year` and `balance` are defined **outside** the loop.
- This means `year` and `balance` are used for all iterations.
- They will be also used outside of the loop.

The `while` Statement

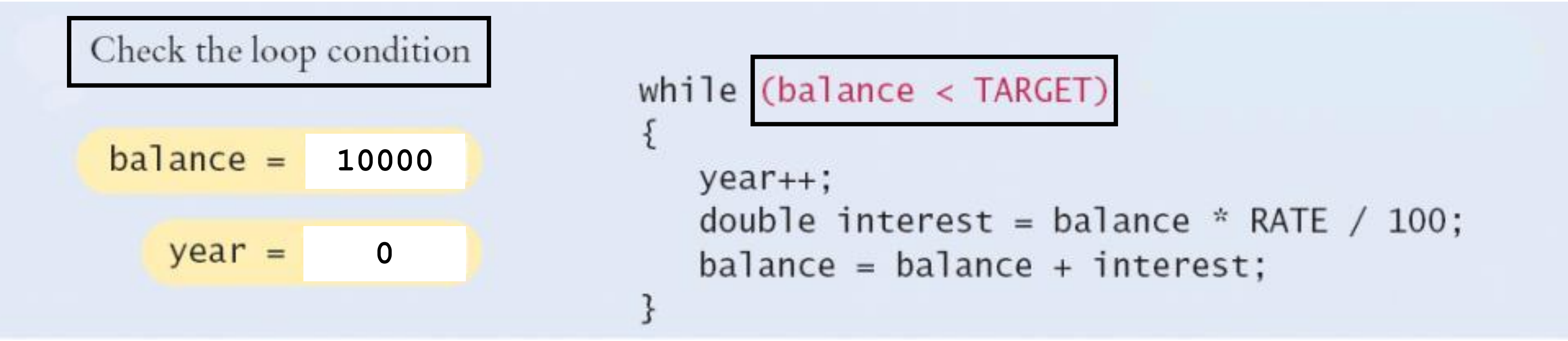

This variable is defined outside the loop and updated in the loop.
If the condition never becomes false, an infinite loop occurs.
Beware of "off-by-one" errors in the loop condition.
Don't put a semicolon here!
These statements are executed while the condition is true.
This variable is created in each loop iteration.
Braces are not required if the body contains a single statement, but it's good to always use them.
Lining up braces is a good idea.

```
double balance = 0;
.
.
.
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

The Complete Program

```
int main()
{
   const double RATE = 5;
   const double INITIAL_BALANCE = 10000;
   const double TARGET = 2 * INITIAL_BALANCE;

   double balance = INITIAL_BALANCE;
   int year = 0;

   while (balance < TARGET)
   {
      year++;
      double interest = balance * RATE / 100;
      balance = balance + interest;
   }

   cout << "The investment doubled after "
        << year << " years." << endl;

   return 0;
}
```

Program Run

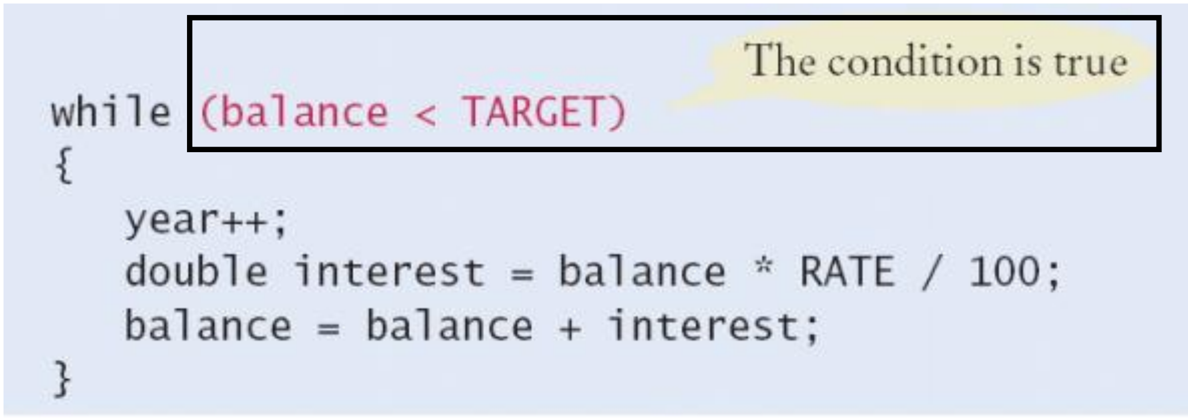


Check the loop condition

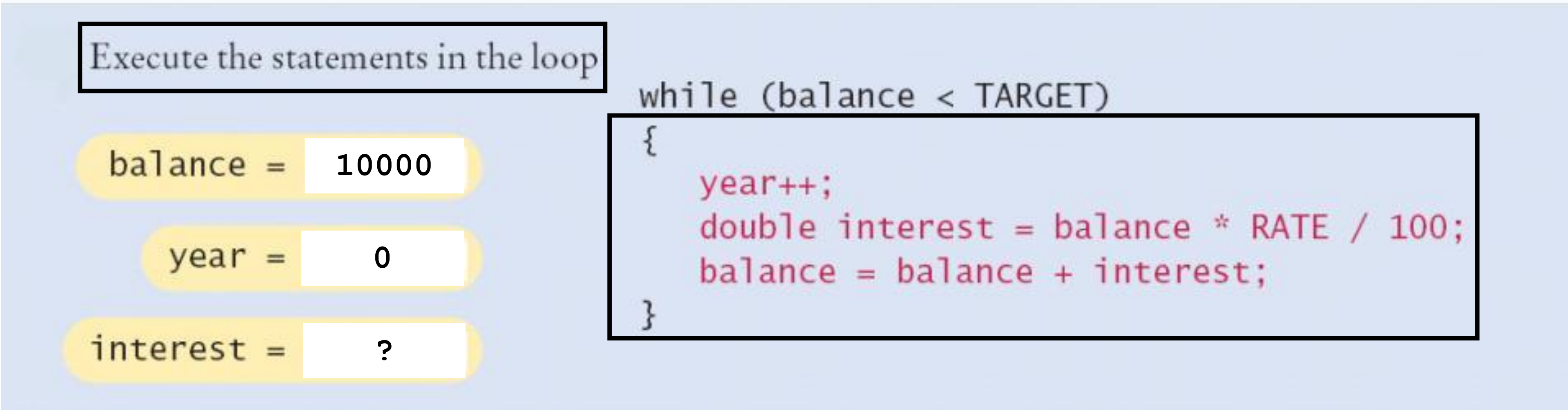
```
balance = 10000

   year = 0
```

```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

Program Run

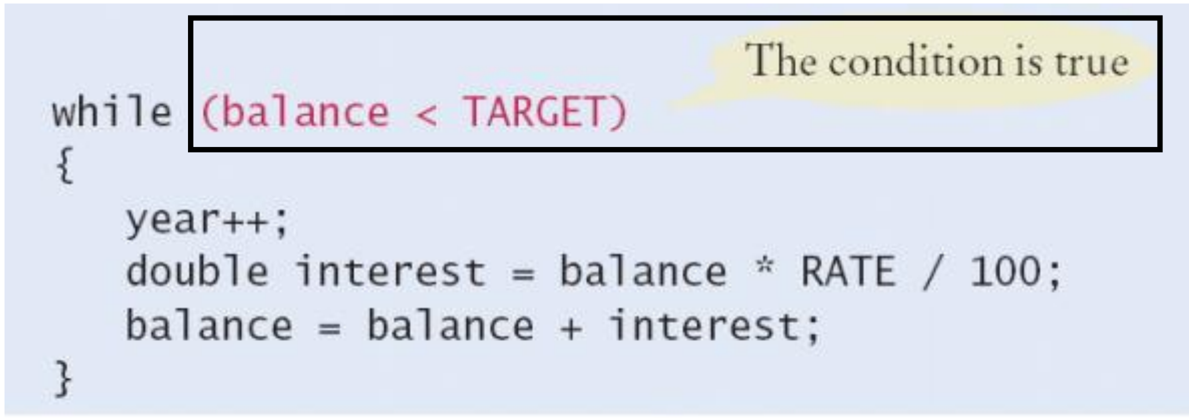


```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

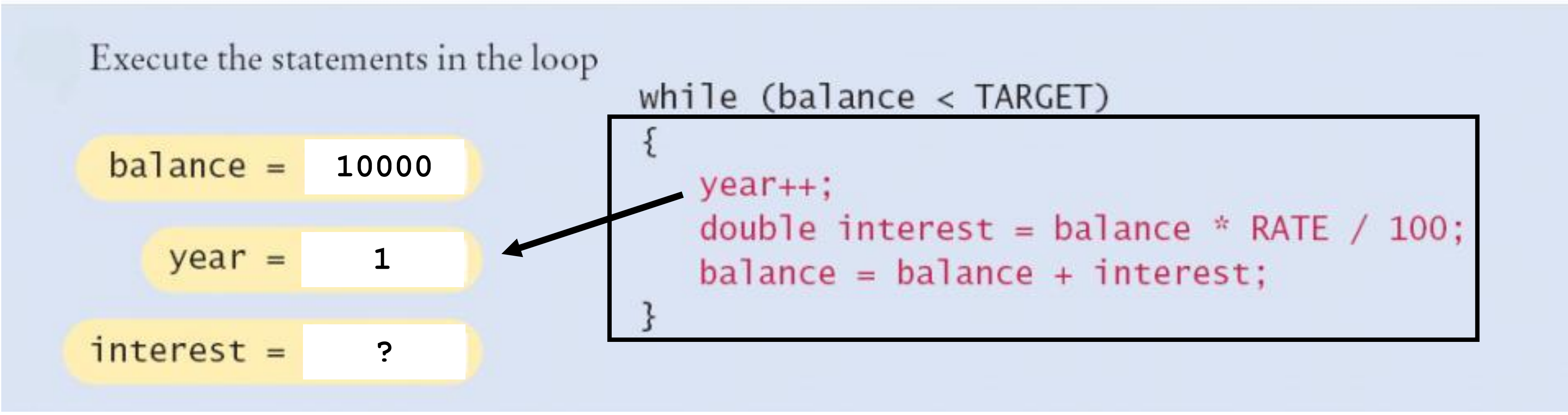


```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

Program Run

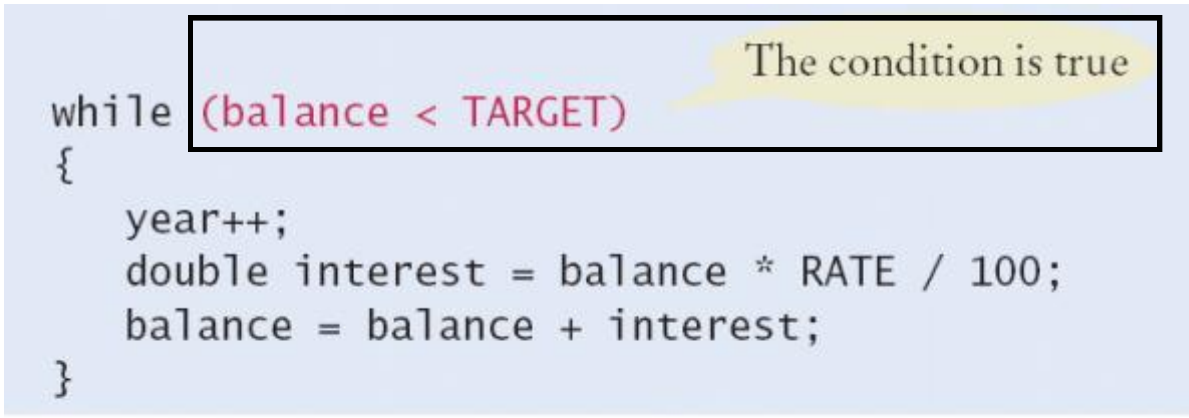


```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

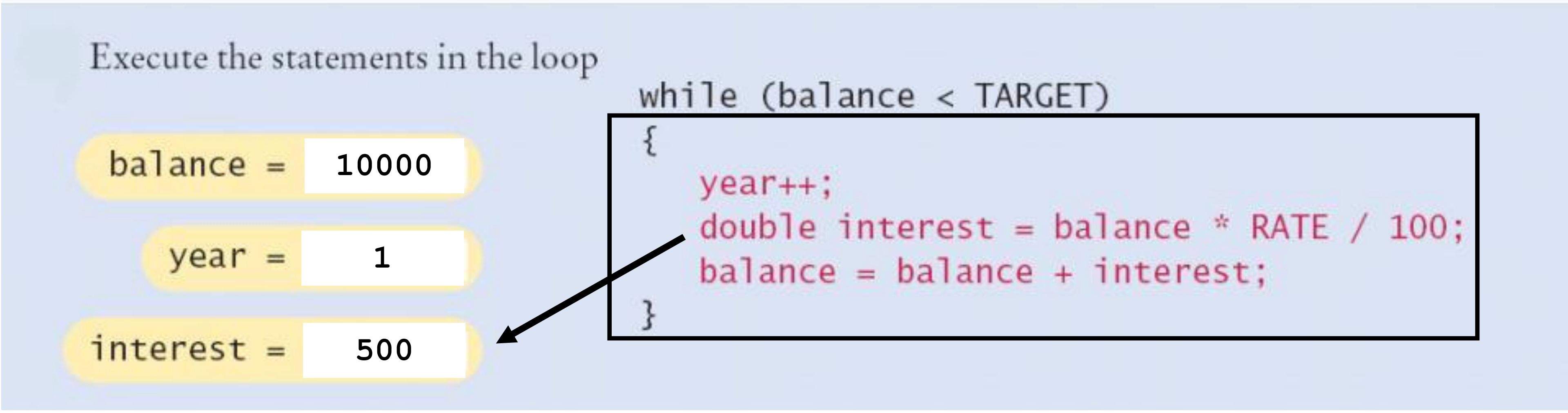


```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

Program Run

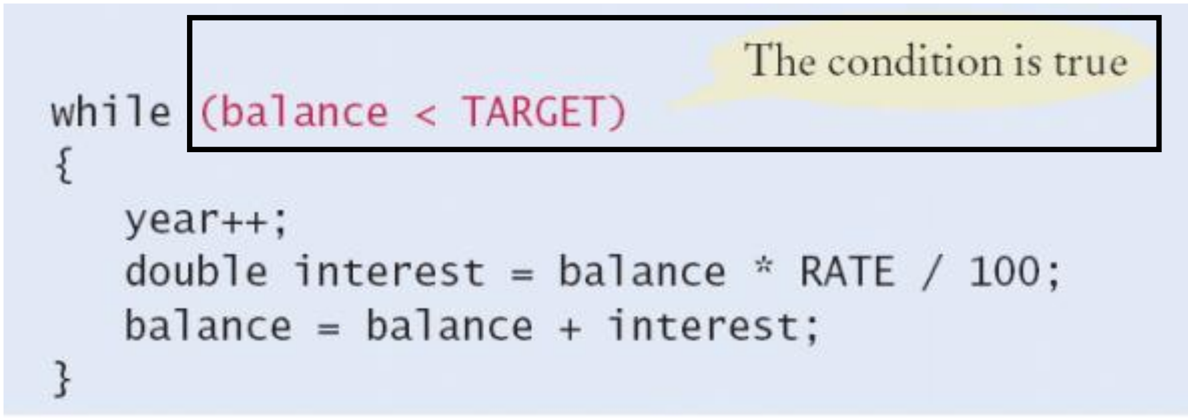


```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

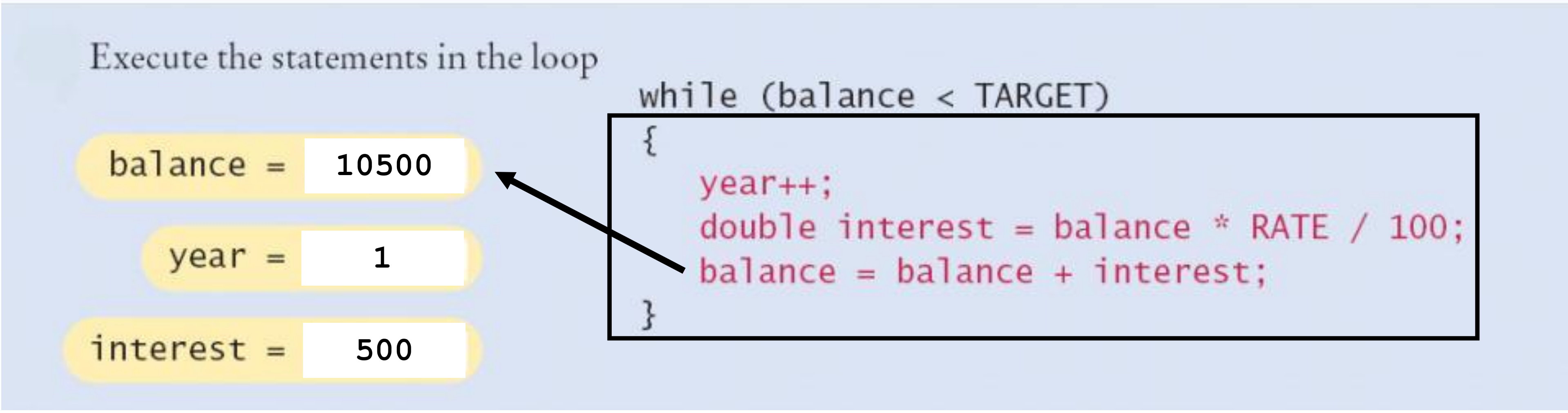


```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

Program Run

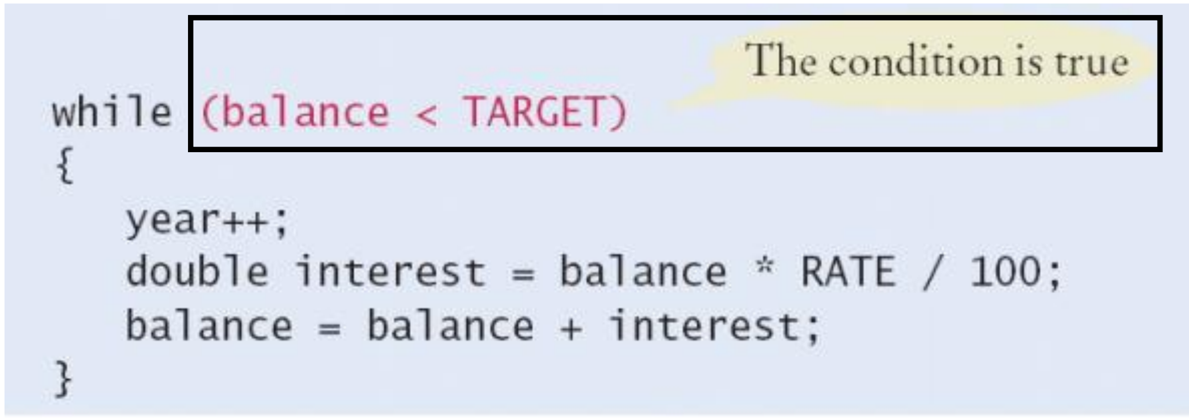


```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

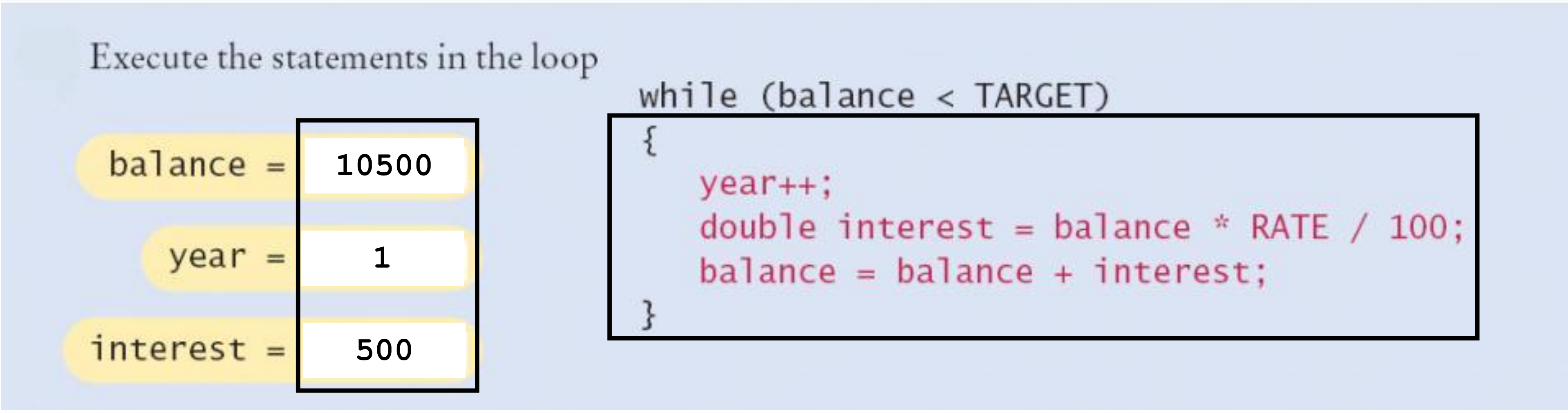


```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

Program Run

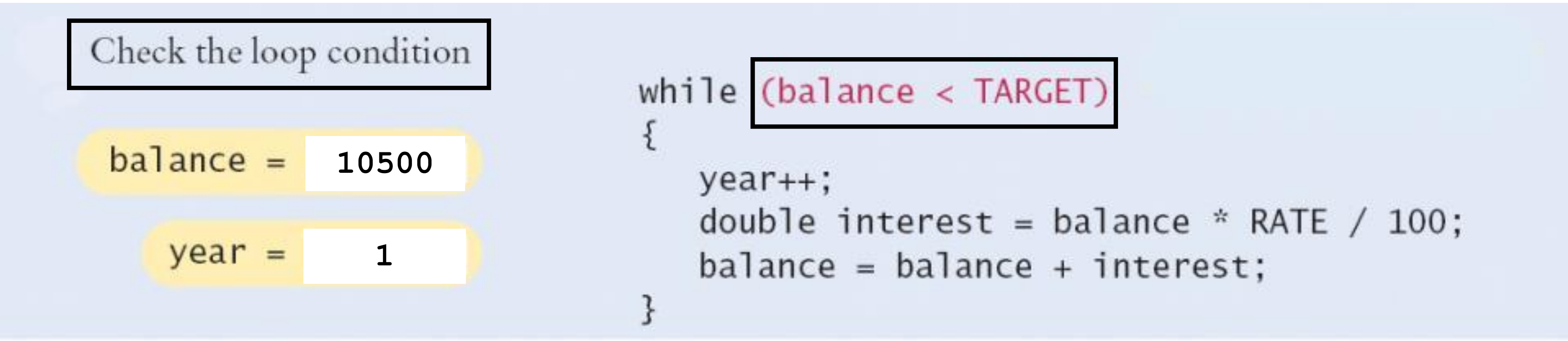


```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```



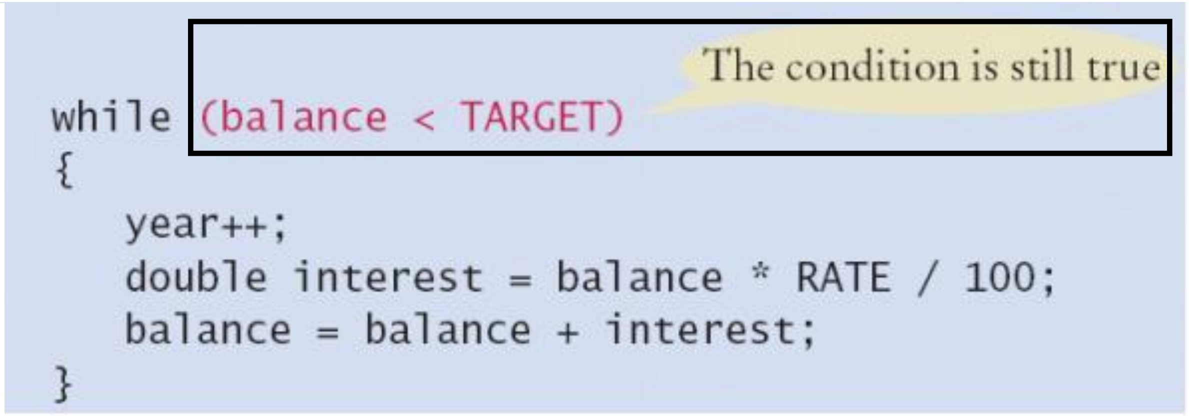
```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

Program Run

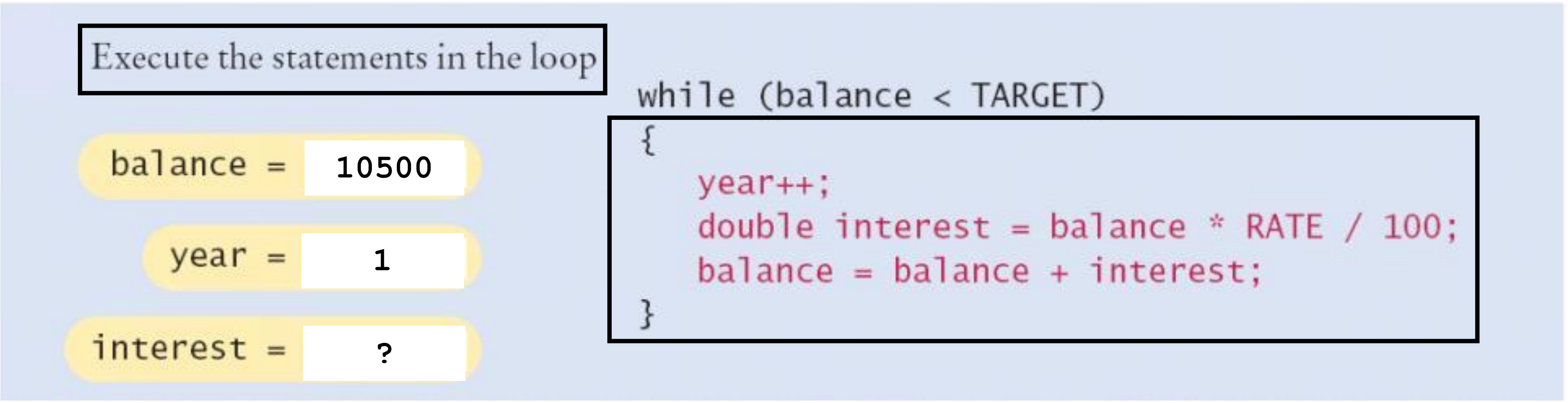


```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

Program Run

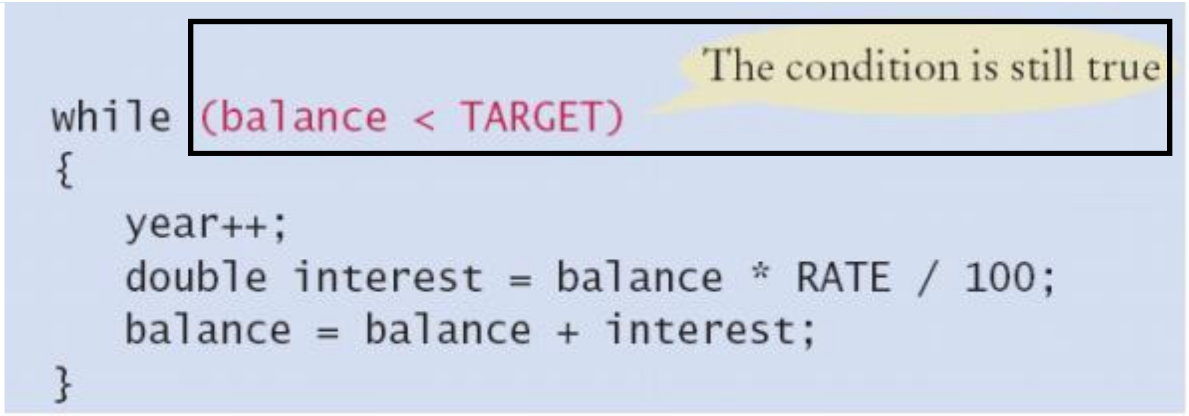


```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

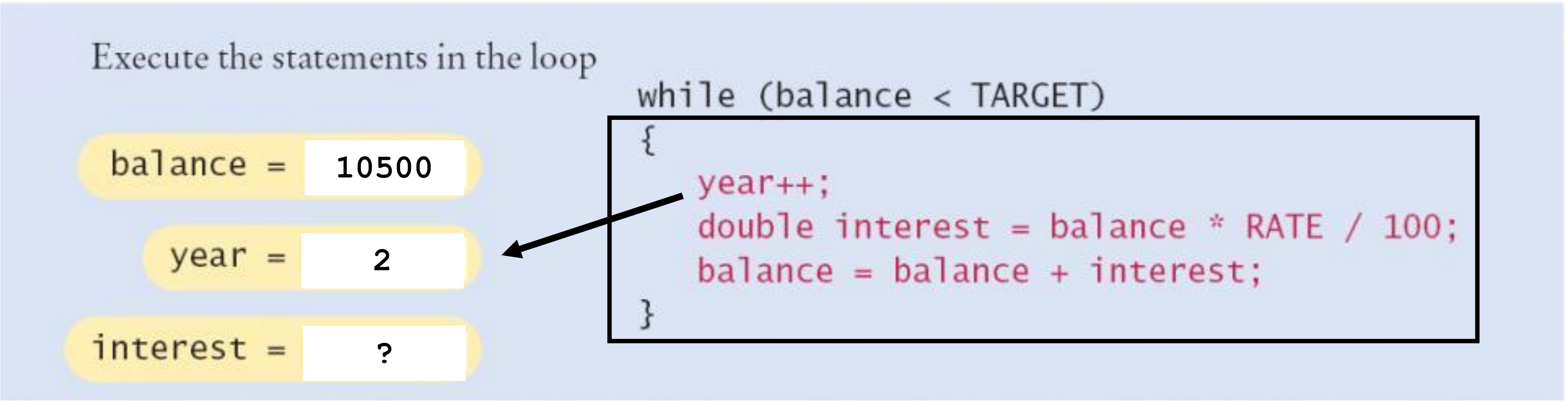


```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

Program Run



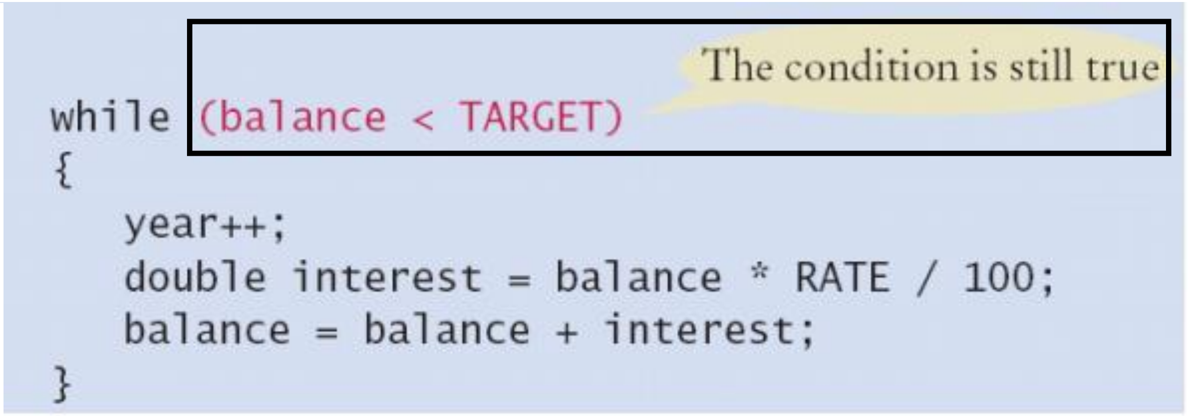
```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```



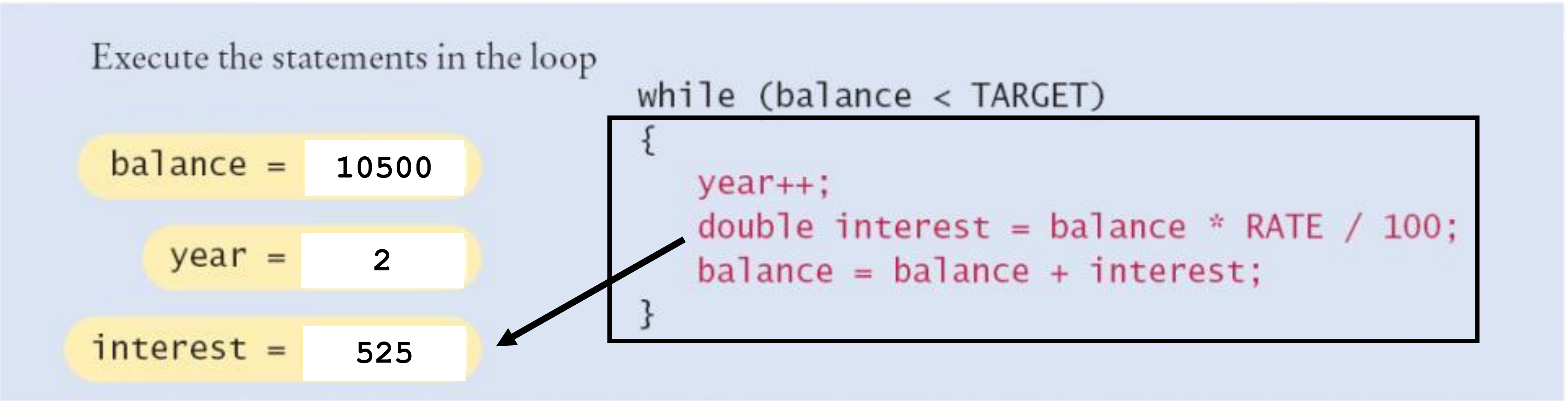
Execute the statements in the loop

```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

Program Run

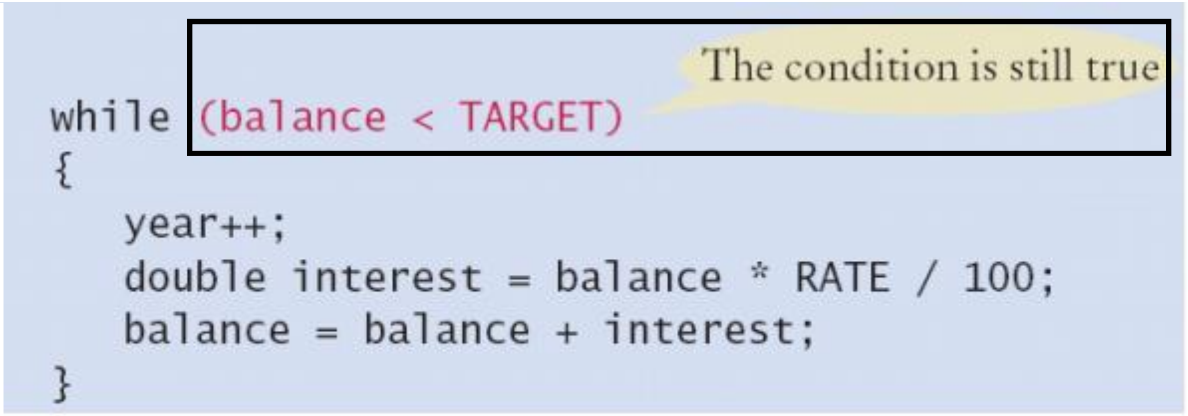


```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

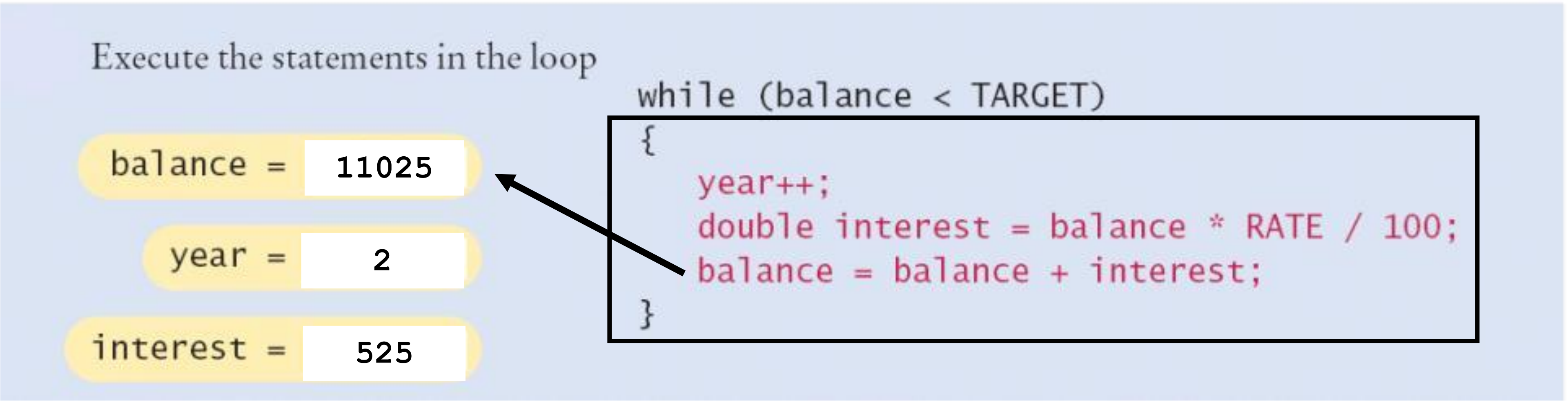


```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

Program Run

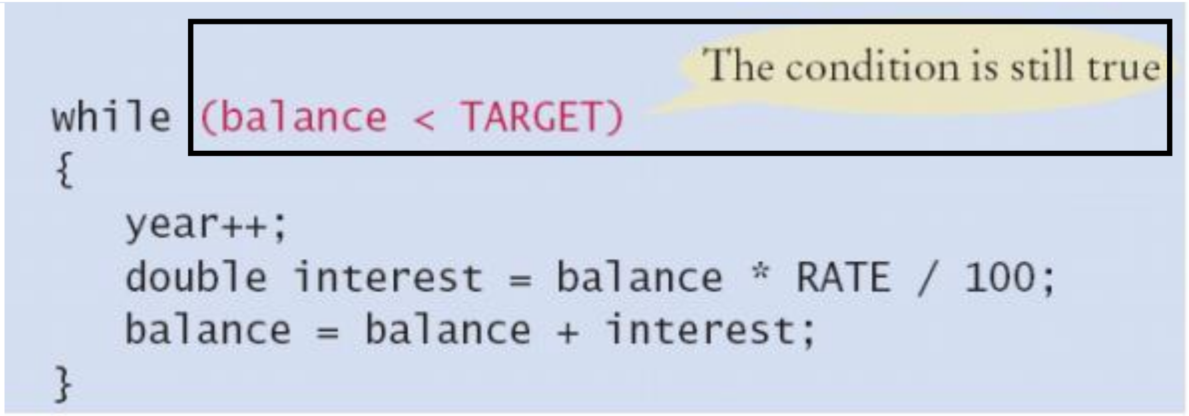


```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

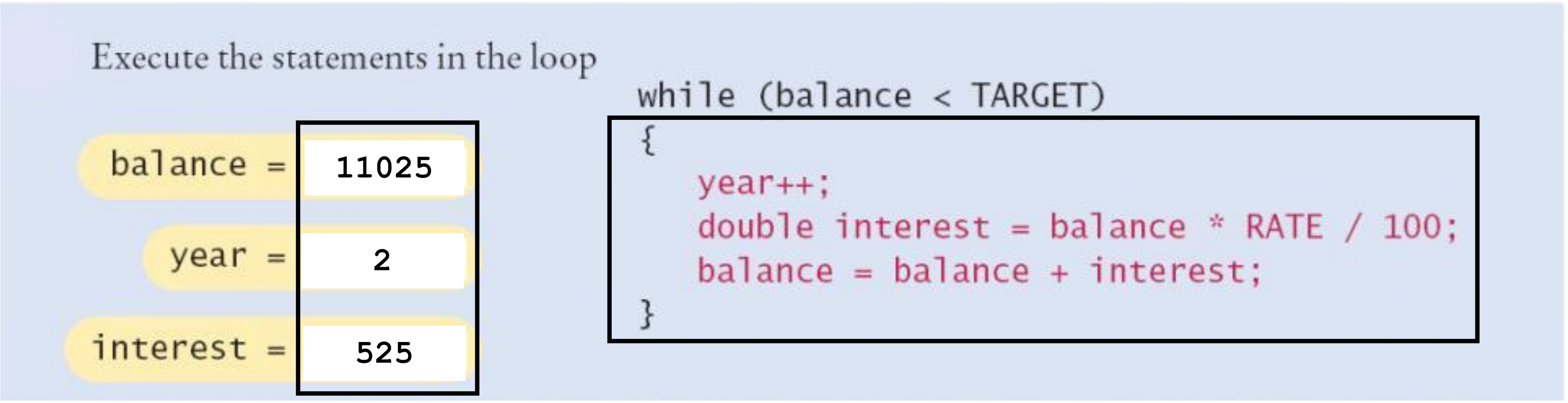


```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

Program Run

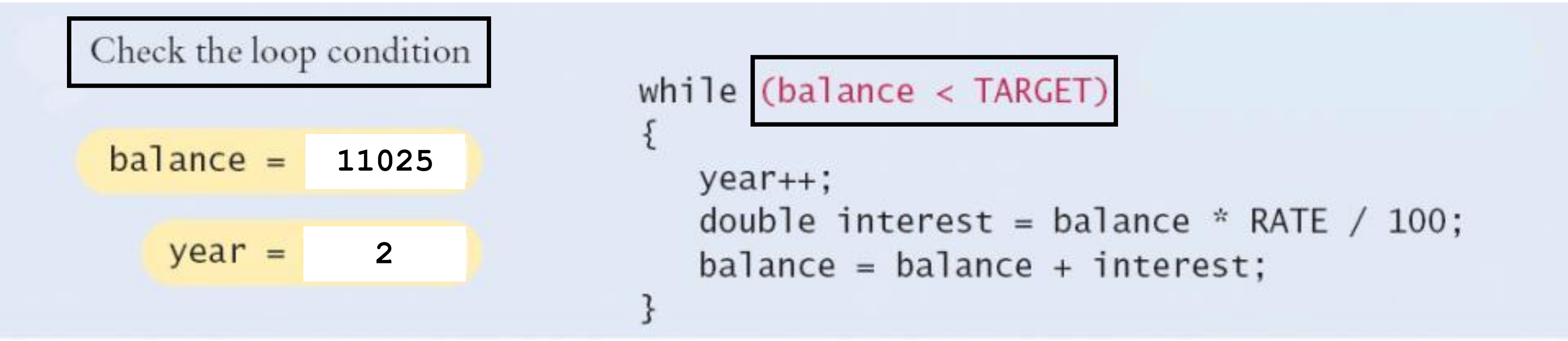


```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```



```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

Program Run



```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

Program Run

before entering while's body		at the end of while's body		
`balance`	`year`	`interest`	`balance`	`year`
10000.00	0	500.00	10500.00	1
10500.00	1	525.00	11025.00	2

…this process goes on and on …

Program Run

before entering while's body		at the end of while's body		
`balance`	`year`	`interest`	`balance`	`year`
10000.00	0	500.00	10500.00	1
10500.00	1	525.00	11025.00	2
11025.00	2	551.25	11576.25	3

…this process goes on and on …

Program Run

…this process goes on and on …

before entering while's body		at the end of while's body		
`balance`	`year`	`interest`	`balance`	`year`
10000.00	0	500.00	10500.00	1
10500.00	1	525.00	11025.00	2
11025.00	2	551.25	11576.25	3
11576.25	3	578.81	12155.06	4
12155.06	4	607.75	12762.82	5

Program Run

…this process goes on and on …

before entering while's body		at the end of while's body		
`balance`	`year`	`interest`	`balance`	`year`
10000.00	0	500.00	10500.00	1
10500.00	1	525.00	11025.00	2
11025.00	2	551.25	11576.25	3
11576.25	3	578.81	12155.06	4
12155.06	4	607.75	12762.82	5
12762.82	5	638.14	13400.96	6
13400.96	6	670.05	14071.00	7
14071.00	7	703.55	14774.55	8
14774.55	8	738.73	15513.28	9

Program Run

…this process goes on and on …

before entering while's body		at the end of while's body		
`balance`	`year`	`interest`	`balance`	`year`
10000.00	0	500.00	10500.00	1
10500.00	1	525.00	11025.00	2
11025.00	2	551.25	11576.25	3
11576.25	3	578.81	12155.06	4
12155.06	4	607.75	12762.82	5
12762.82	5	638.14	13400.96	6
13400.96	6	670.05	14071.00	7
14071.00	7	703.55	14774.55	8
14774.55	8	738.73	15513.28	9
15513.28	9	775.66	16288.95	10
16288.95	10	814.45	17103.39	11
17103.39	11	855.17	17958.56	12
17958.56	12	897.93	18856.49	13
18856.49	13	942.82	19799.32	14

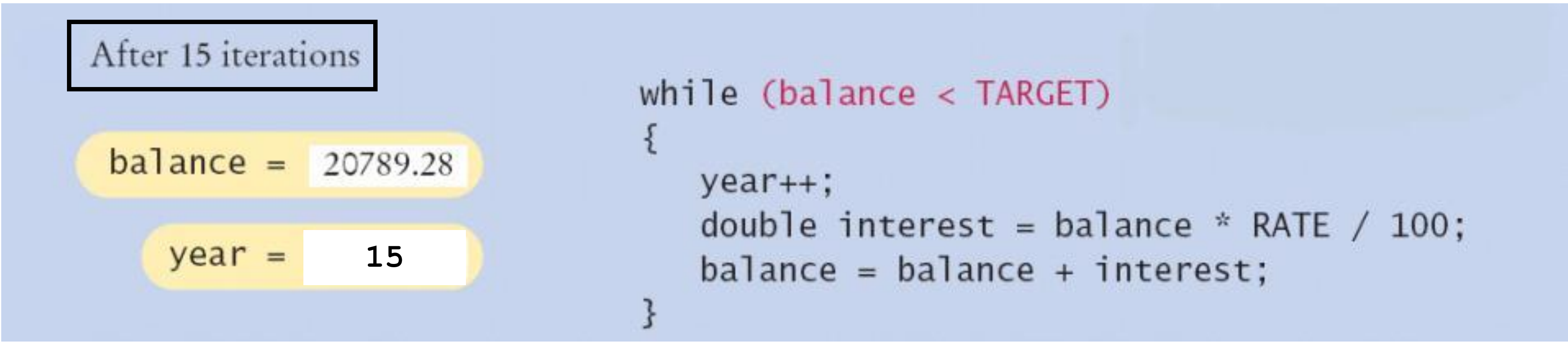
Program Run

…this process goes on and on …

…until the **`balance`** is finally(!) over $20,000 and the test becomes **`false`**.

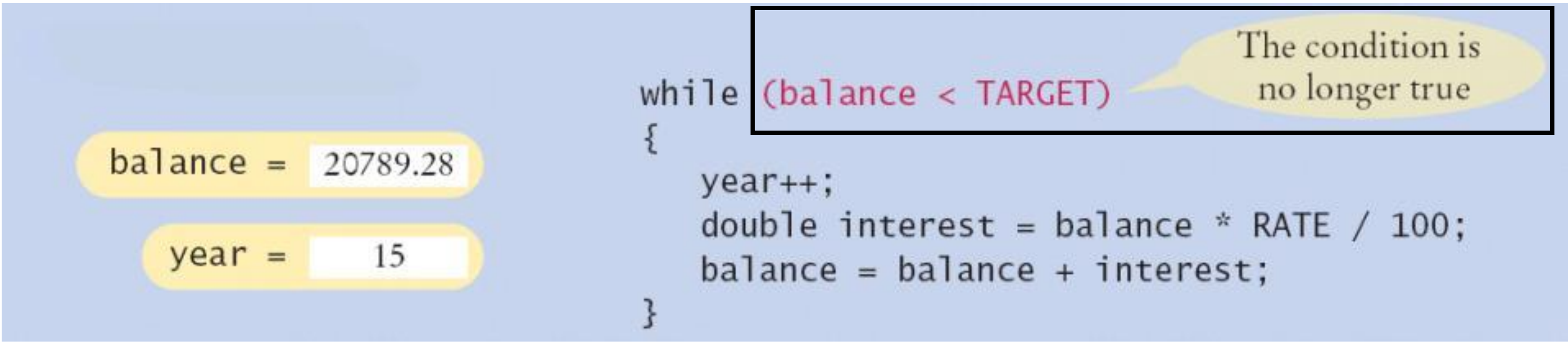
before entering while's body		at the end of while's body		
`balance`	`year`	`interest`	`balance`	`year`
10000.00	0	500.00	10500.00	1
10500.00	1	525.00	11025.00	2
11025.00	2	551.25	11576.25	3
11576.25	3	578.81	12155.06	4
12155.06	4	607.75	12762.82	5
12762.82	5	638.14	13400.96	6
13400.96	6	670.05	14071.00	7
14071.00	7	703.55	14774.55	8
14774.55	8	738.73	15513.28	9
15513.28	9	775.66	16288.95	10
16288.95	10	814.45	17103.39	11
17103.39	11	855.17	17958.56	12
17958.56	12	897.93	18856.49	13
18856.49	13	942.82	19799.32	14
19799.32	14	989.97	20789.28	15
		while statement is over		

Program Run



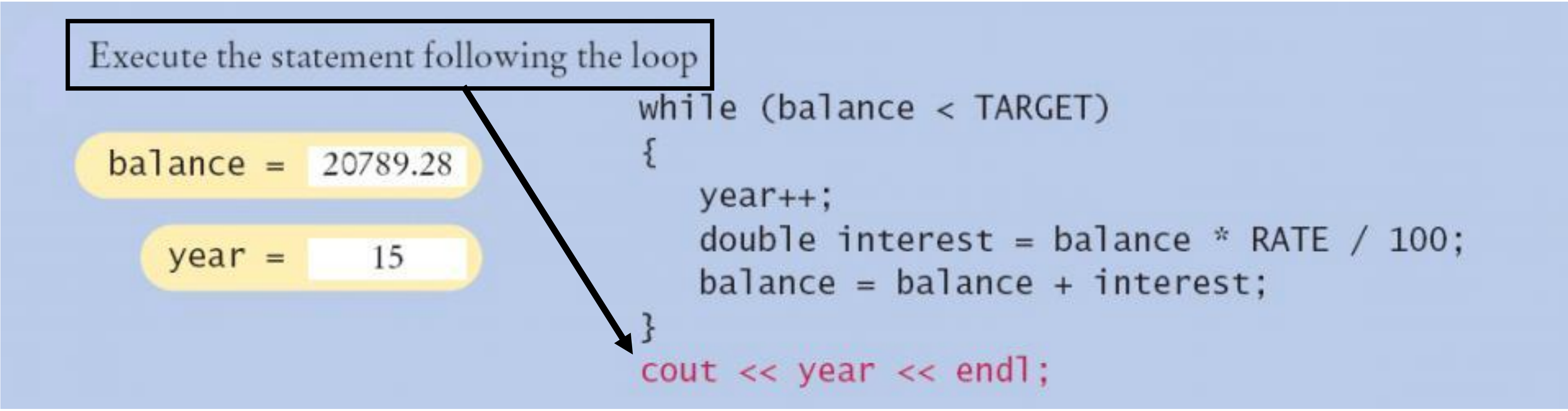
```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

Program Run



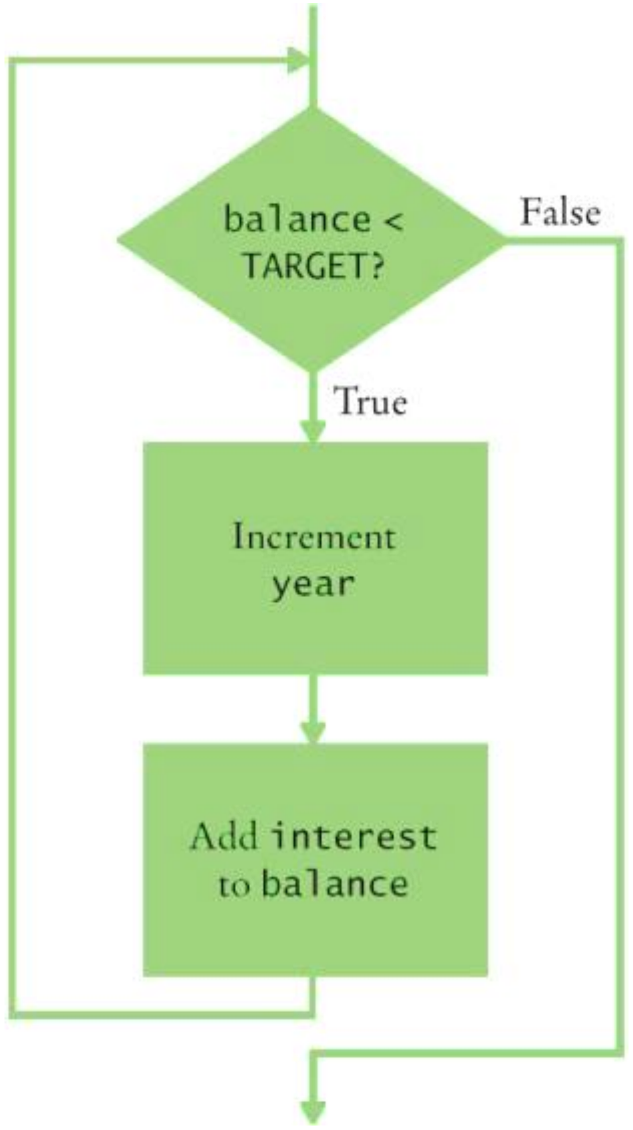
```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

Program Run



```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
cout << year << endl;
```

Flowchart


balance < TARGET?
False
True
Increment year
Add interest to balance

More `while` Examples

For each of the following, do a hand-trace

Example of Normal Execution

while loop to hand-trace

```
1 i = 3;
2 while (i > 0)
  {
3    cout << i << " ";
4    i--;
  }
5 return 0
```

trace

line	i	comment
1	3	
2	"	condition is true
3	"	print "3 "
4	2	

Example of Normal Execution

while loop to hand-trace

```
1 i = 3;
2 while (i > 0)
  {
3    cout << i << " ";
4    i--;
  }
5 return 0
```

trace

line	i	comment
1	3	
2	"	condition is true
3	"	print "3 "
4	2	
2	"	condition is true
3	"	print "2 "
4	1	

Example of Normal Execution

while loop to hand-trace

```
1 i = 3;
2 while (i > 0)
  {
3    cout << i << " ";
4    i--;
  }
5 return 0
```

trace

line	i	comment
1	3	
2	"	condition is true
3	"	print "3 "
4	2	
2	"	condition is true
3	"	print "2 "
4	1	
2	"	condition is true
3	"	print 1
4	0	

Example of Normal Execution

while loop to hand-trace

```
1 i = 3;
2 while (i > 0)
  {
3    cout << i << " ";
4    i--;
  }
5 return 0
```

trace

line	i	comment
1	3	
2	“	condition is true
3	“	print “3 “
4	2	
2	“	condition is true
3	“	print “2 “
4	1	
2	“	condition is true
3	“	print 1
4	0	
2	“	condition is false
5	“	end of program

Example of an Infinite Loop

while loop to hand-trace

```
1 i = 3;
2 while (i > 0)
  {
3    cout << i << " ";
4    i++;
  }
5 return 0
```

trace

line	i	comment

Example of an Infinite Loop

while loop to hand-trace

```
1 i = 3;
2 while (i > 0)
  {
3    cout << i << " ";
4    i++;
  }
5 return 0
```

trace

line	i	comment
1	3	
2	“	condition is true
3	“	print “3 “
4	4	
2	“	condition is true
3	“	print “4 “
4	5	
2	“	condition is true
3	“	print “5 “
4	6	
2	“	condition is true
3	“	print “6 “
4	7	

Example of an Infinite Loop

while loop to hand-trace

```
1 i = 3;
2 while (i > 0)
  {
3    cout << i << " ";
4    i++;
  }
5 return 0
```

trace

line	i	comment
1	3	
2	“	condition is true
3	“	print “3 “
4	4	
2	“	condition is true
3	“	print “4 “
4	5	
2	“	condition is true
3	“	print “5 “
4	6	
2	“	condition is true
3	“	print “6 “
4	7	
2	“	condition is true
3	“	print “57“
4	8	
2	“	condition is true
3	“	print “8 “

Example of an Infinite Loop

while loop to hand-trace

```
1 i = 3;
2 while (i > 0)
  {
3    cout << i << " ";
4    i++;
  }
5 return 0
```

- Some time later

trace

4	133	
2	"	condition is true
3	"	print "133 "
4	134	
2	"	condition is true
3	"	print "134 "
4	135	
2	"	condition is true
3	"	print "135 "
4	136	
2	"	condition is true
3	"	print "136 "
4	137	
2	"	condition is true
3	"	print "137"
4	138	
2	"	condition is true
3	"	print "138 "
4	137	

Example of an Infinite Loop

while loop to hand-trace

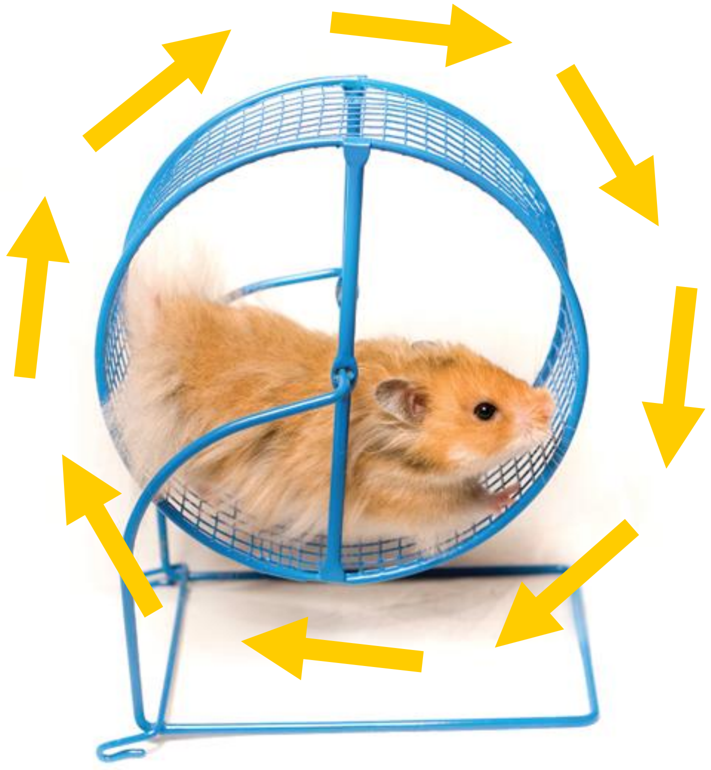
```
1 i = 3;
2 while (i > 0)
  {
3     cout << i << " ";
4     i++;
  }
5 return 0
```

trace

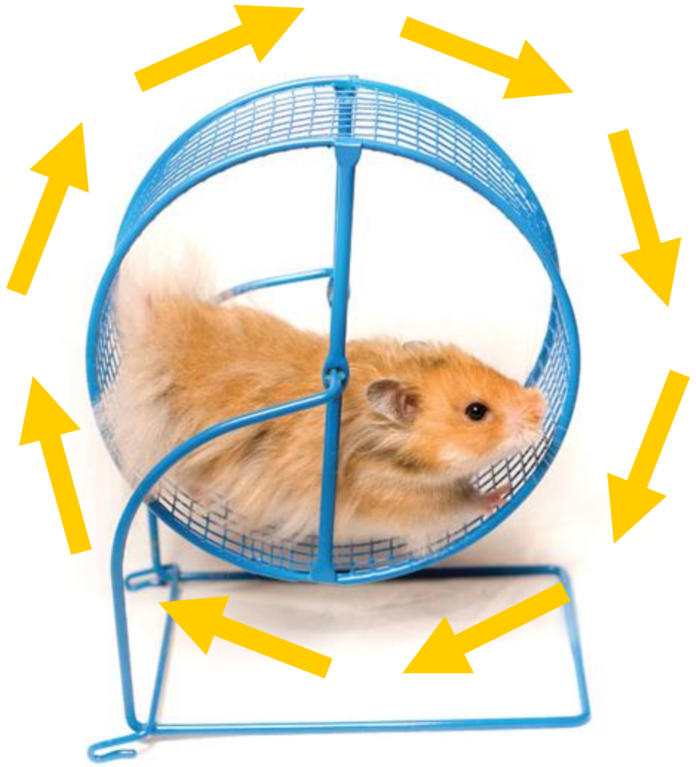
4	429453	
2	“	condition is true
3	“	print “429453 “
4	429454	
2	“	condition is true
3	“	print “429454 “
4	429455	
2	“	condition is true
3	“	print “429455 “
4	429456	
2	“	condition is true
3	“	print “429456 “
4	429457	
2	“	condition is true
3	“	print “429457“
4	429458	
2	“	condition is true
3	“	print “429458 “
4	429457	

- Some more time later
- Will this ever stop?
- Yes, when you get an overflow.
- But that’s not what you want.

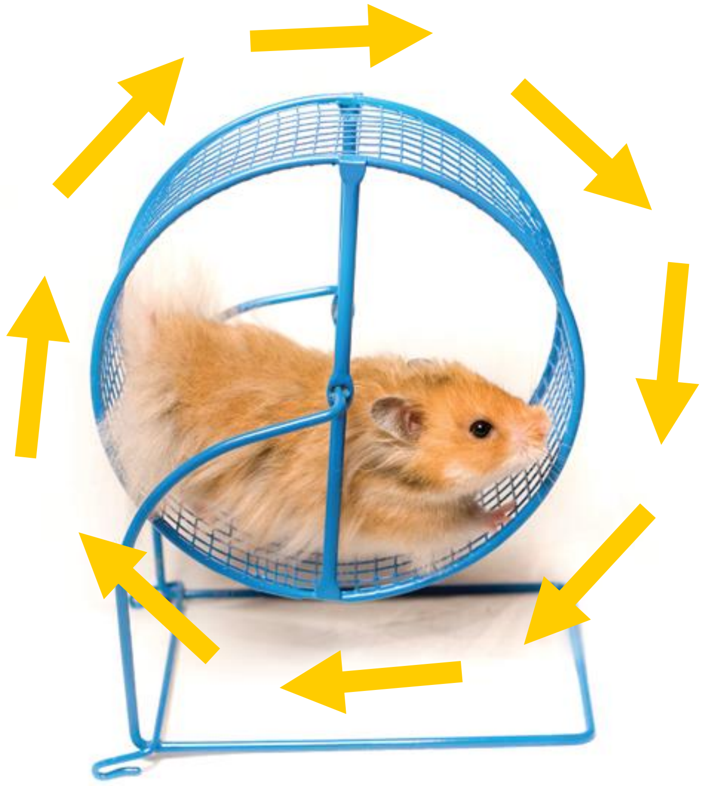
Common Error – Infinite Loops



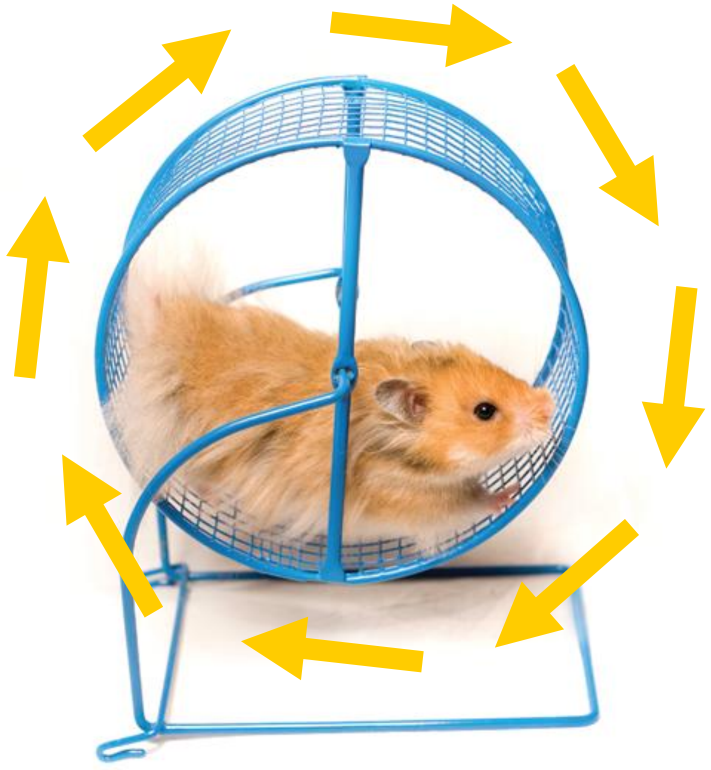
Common Error – Infinite Loops



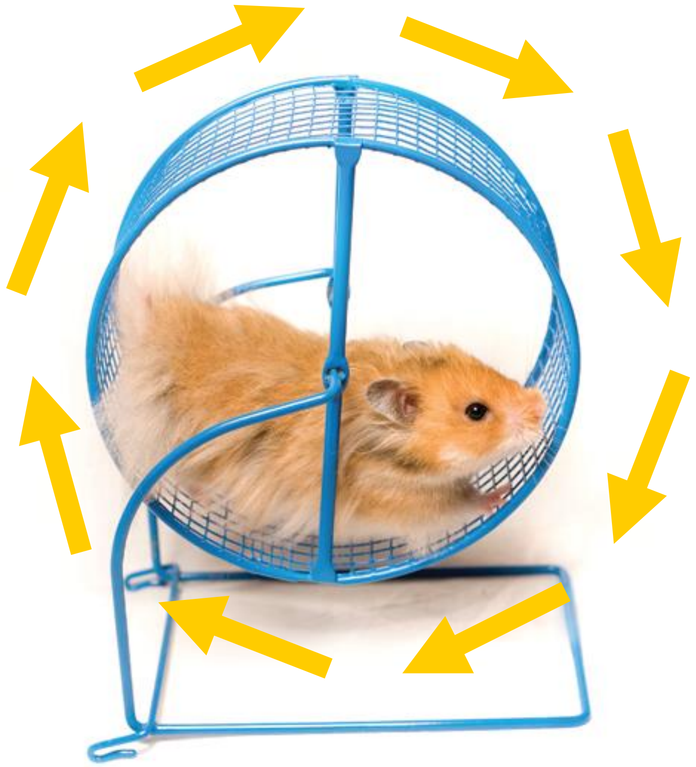
Common Error – Infinite Loops



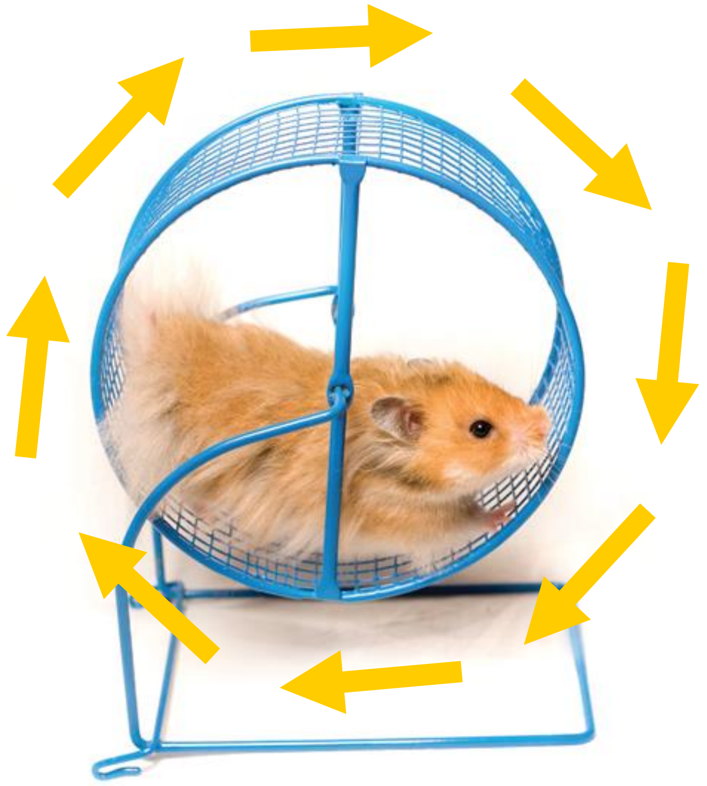
Common Error – Infinite Loops



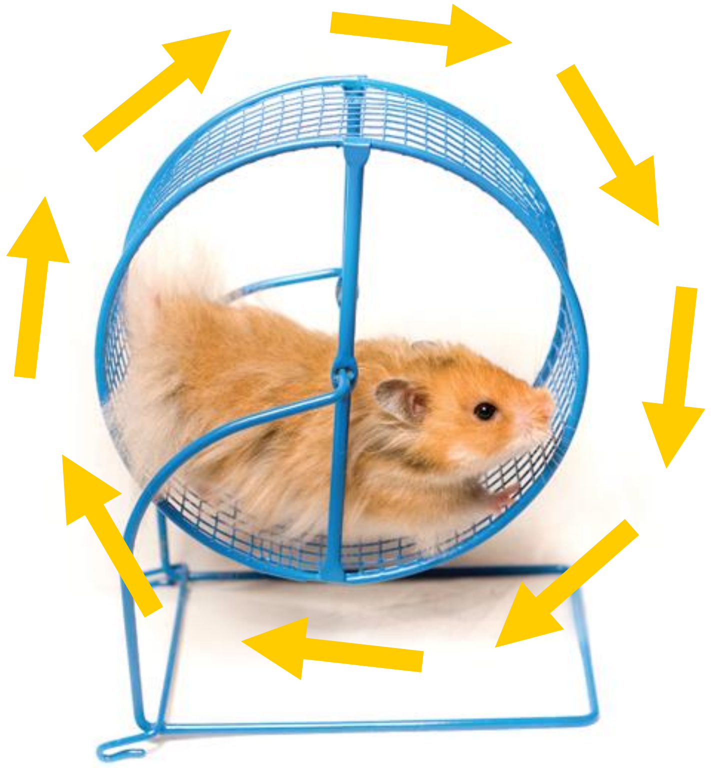
Common Error – Infinite Loops



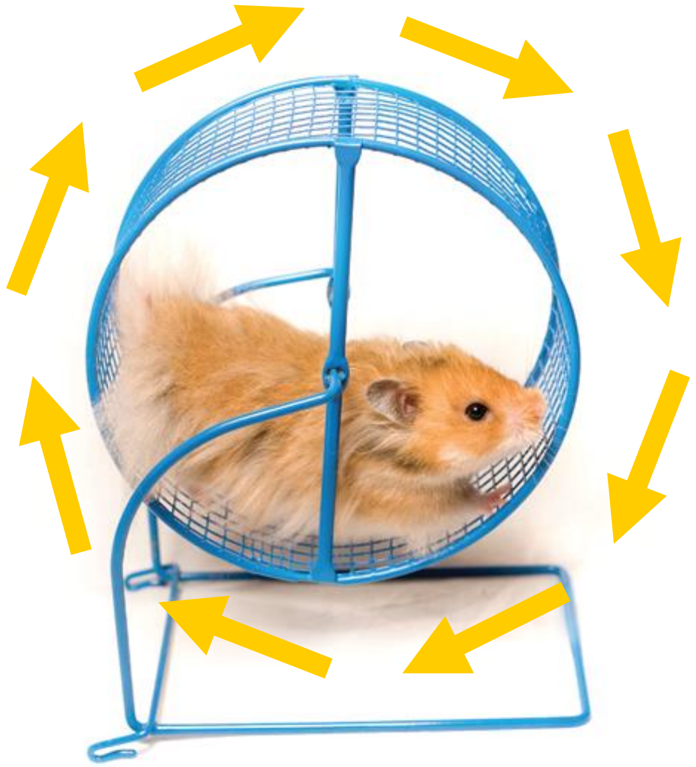
Common Error – Infinite Loops



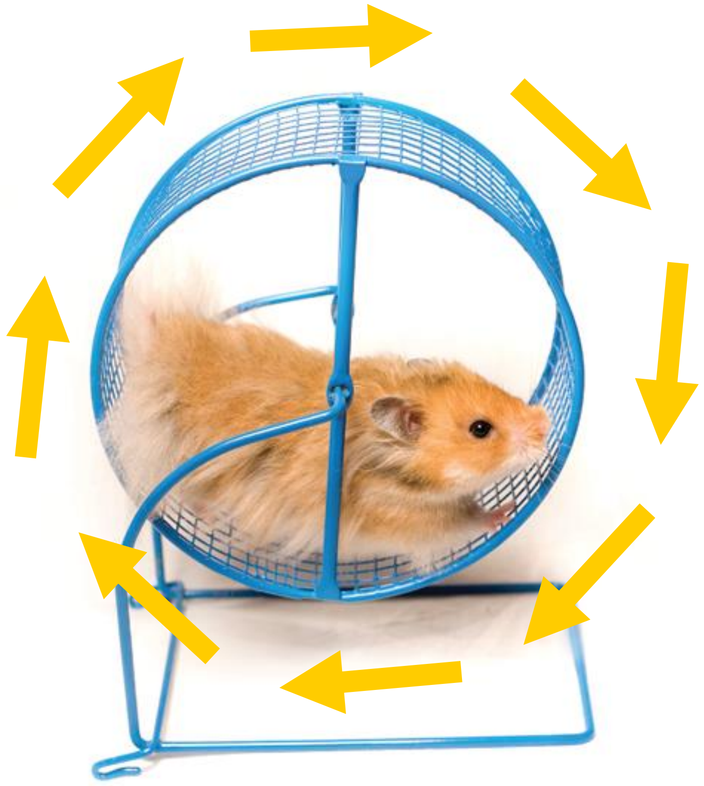
Common Error – Infinite Loops



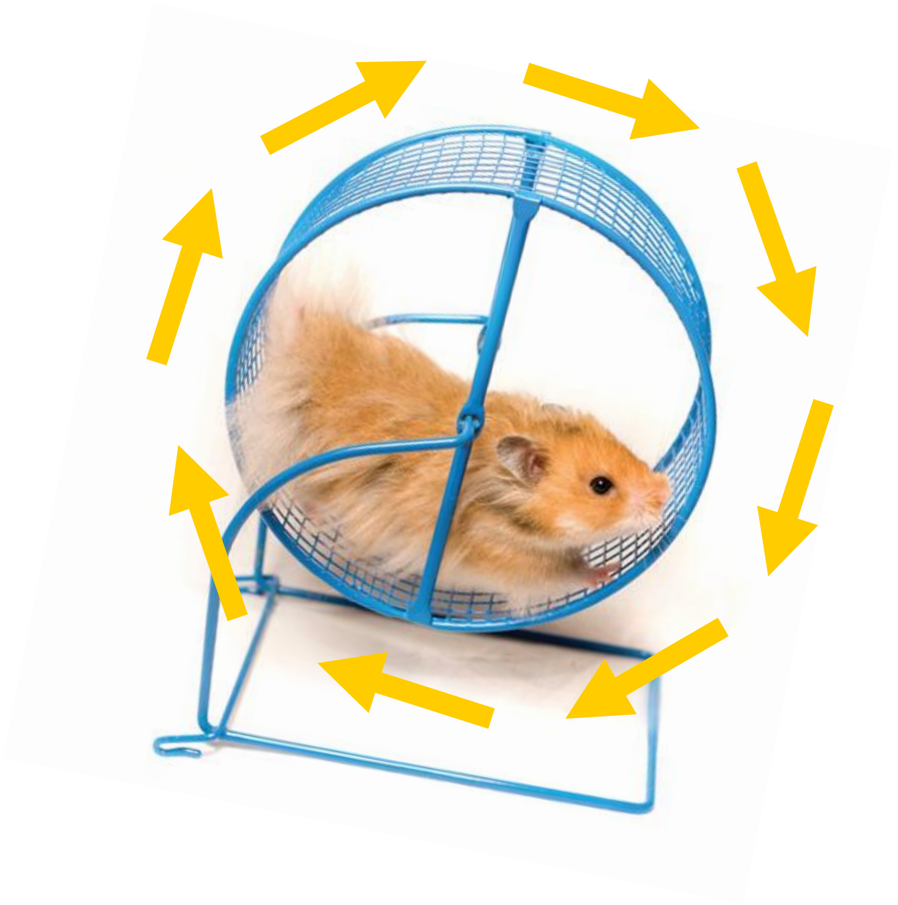
Common Error – Infinite Loops



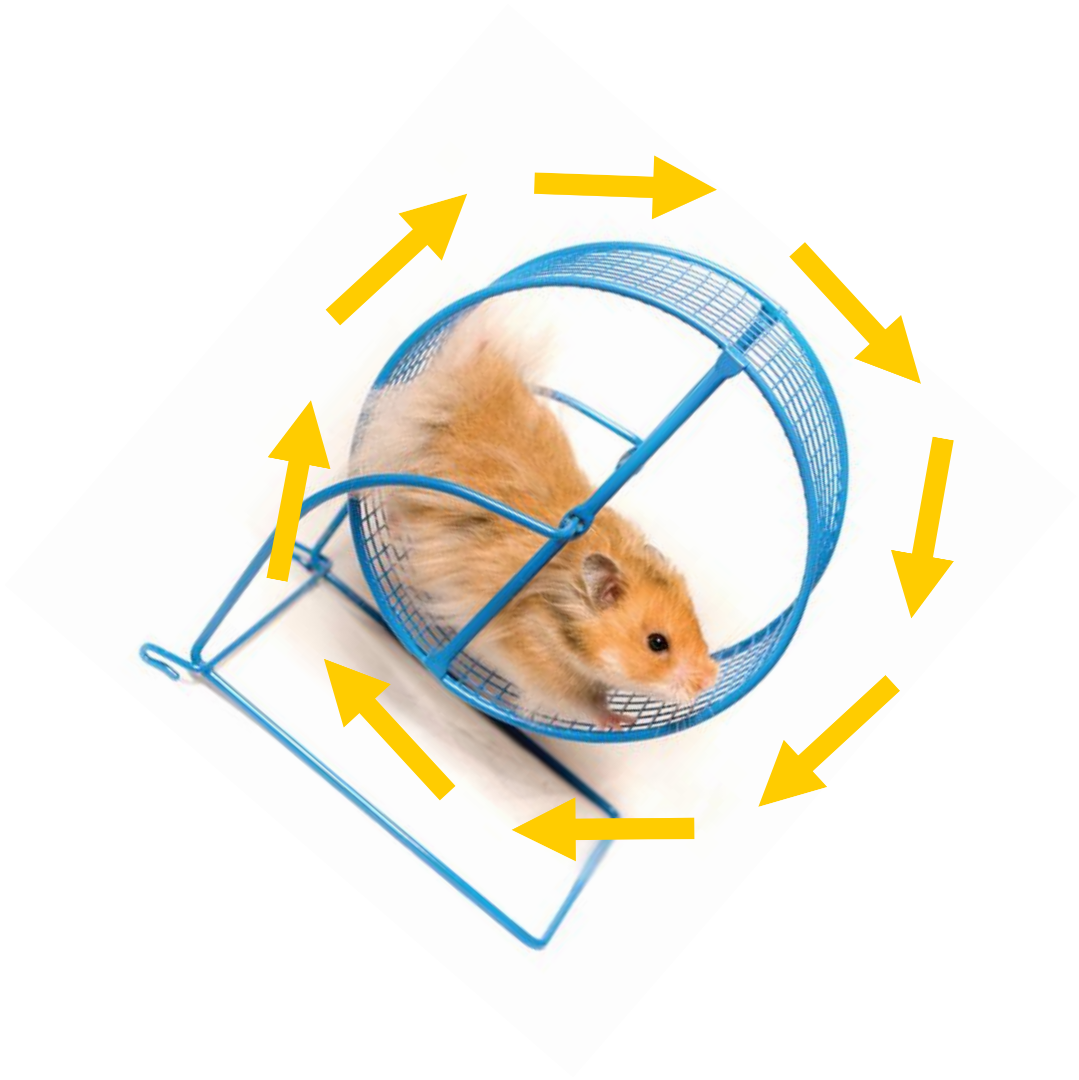
Common Error – Infinite Loops



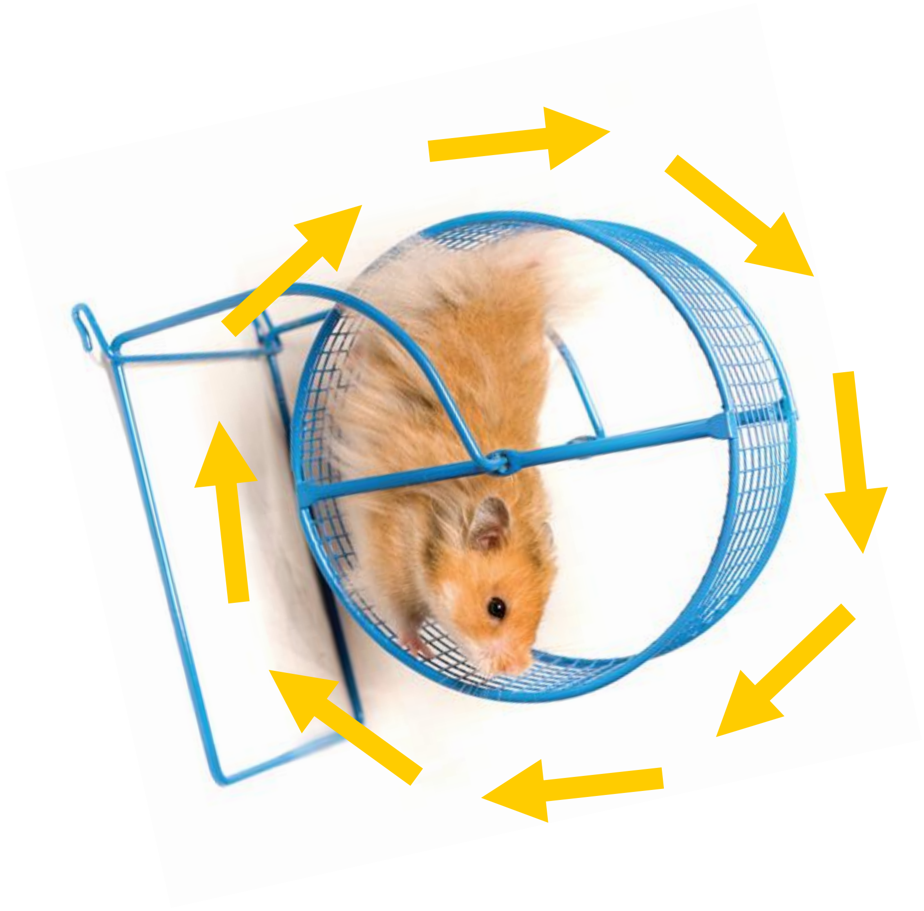
Common Error – Infinite Loops



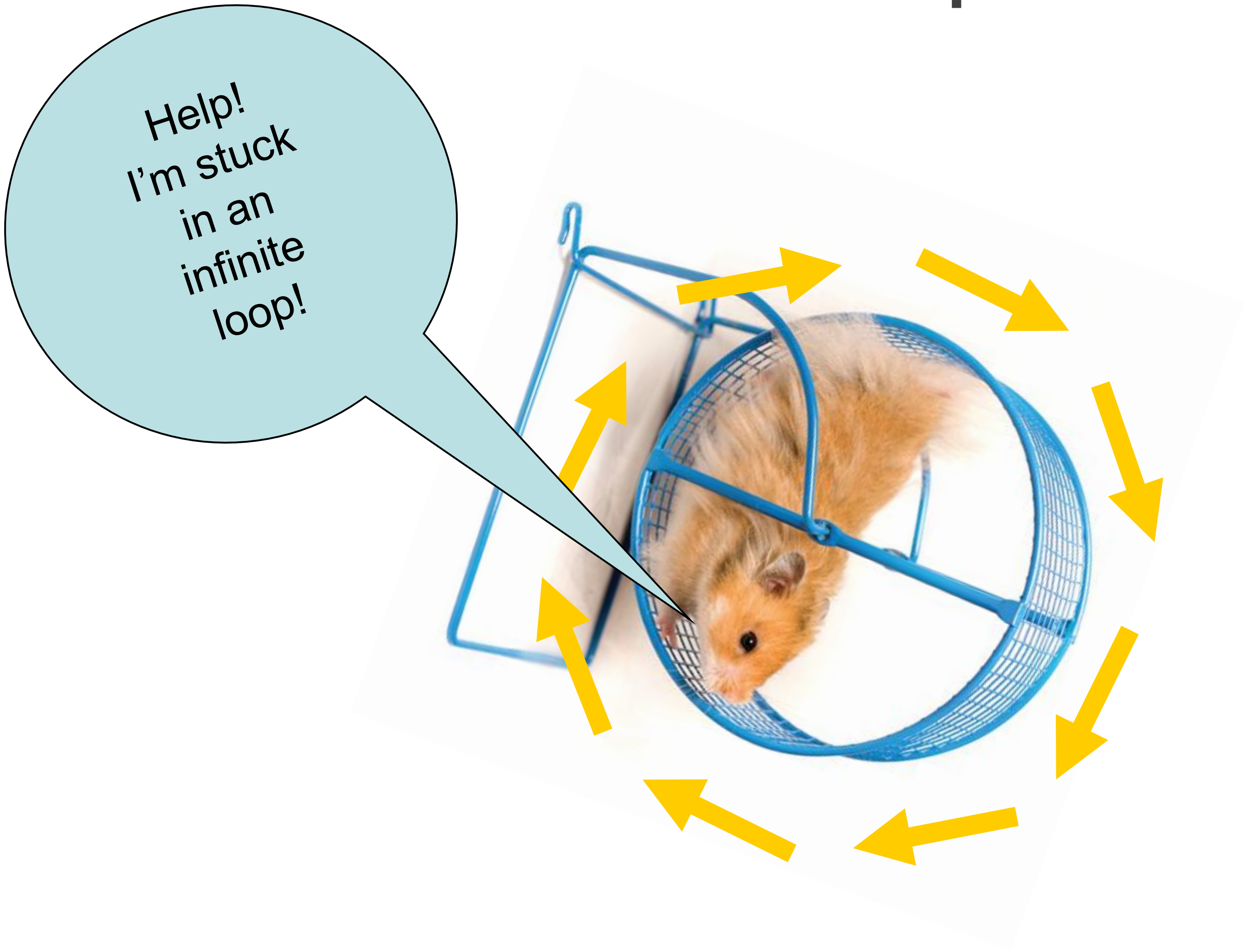
Common Error – Infinite Loops



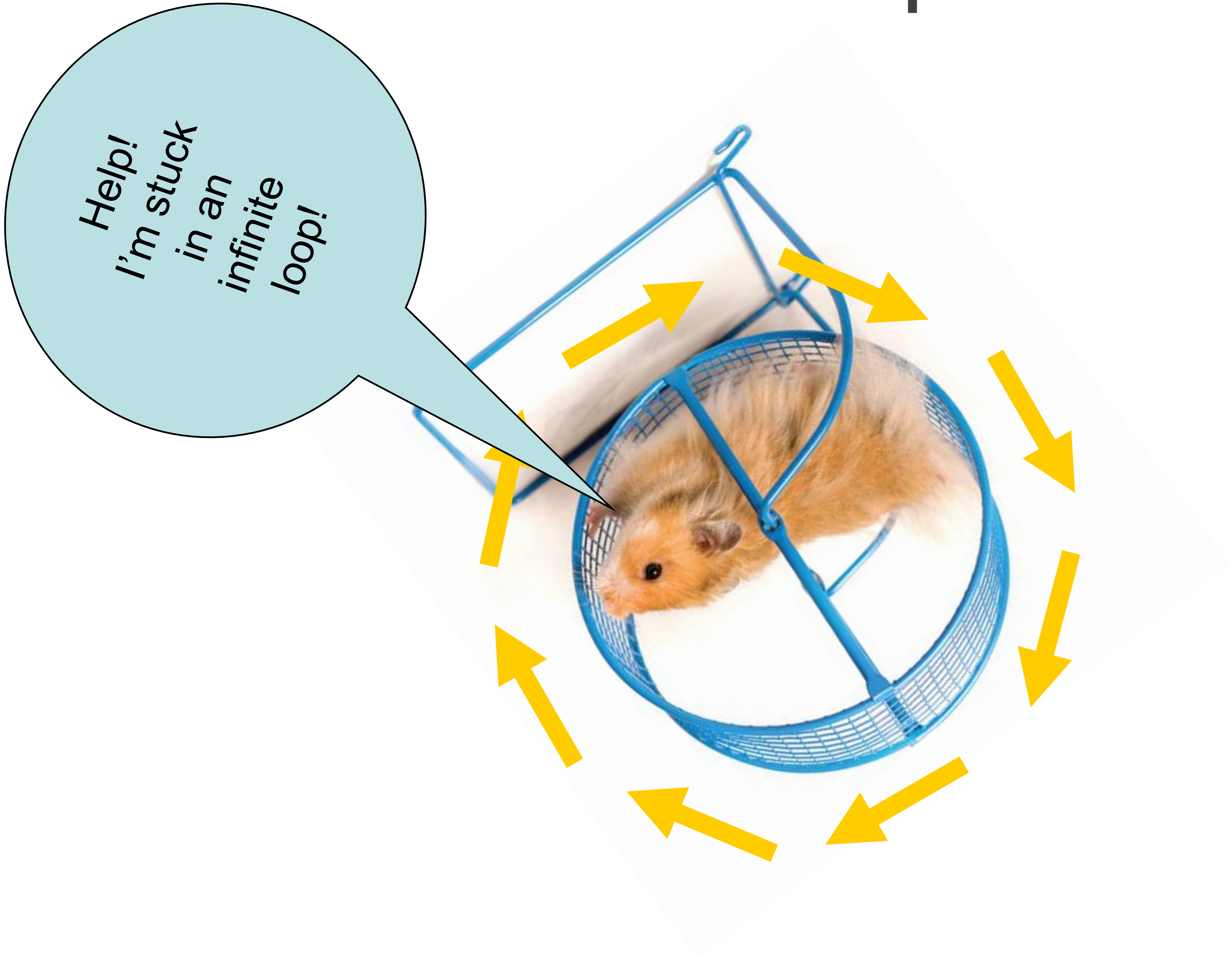
Common Error – Infinite Loops



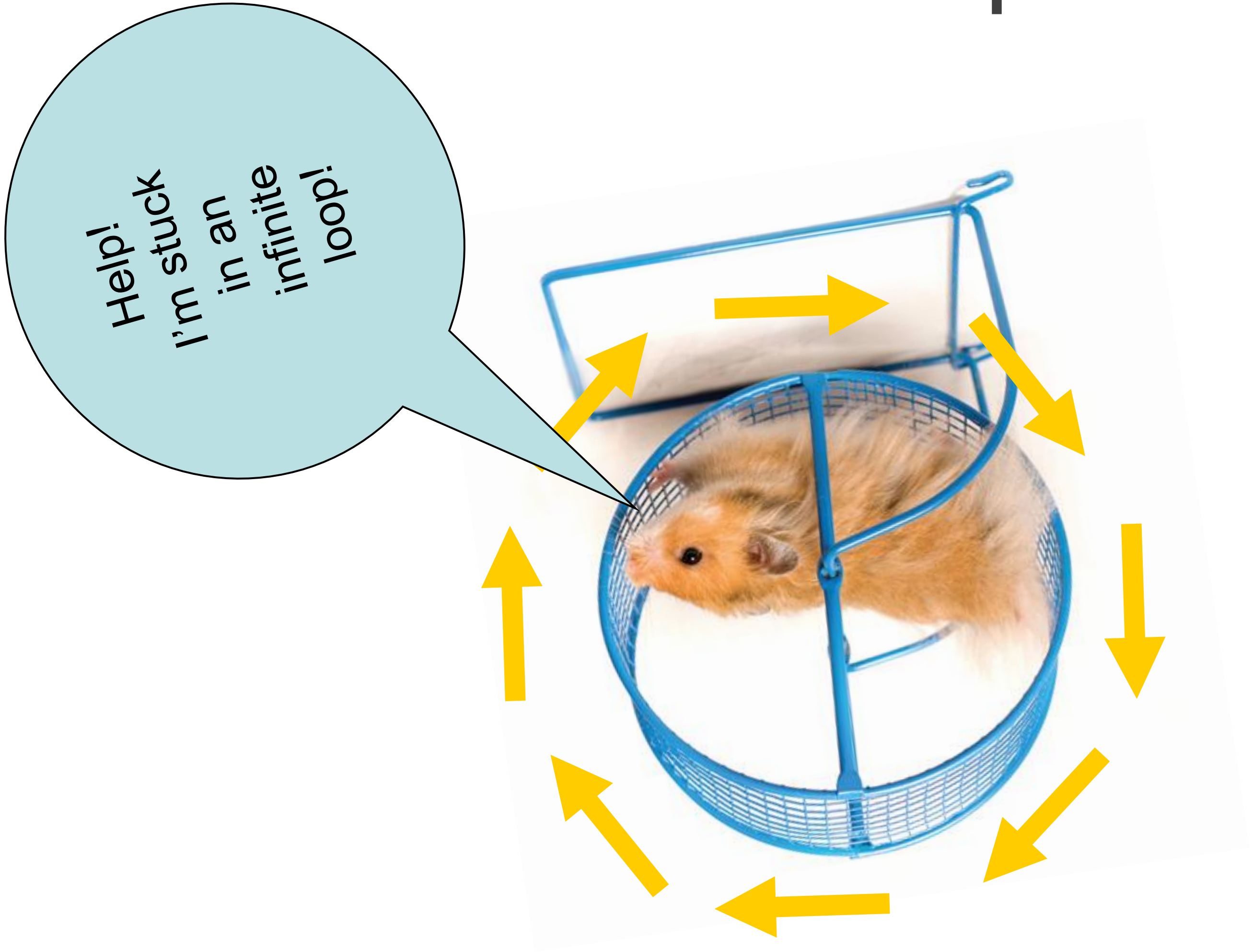
Common Error – Infinite Loops



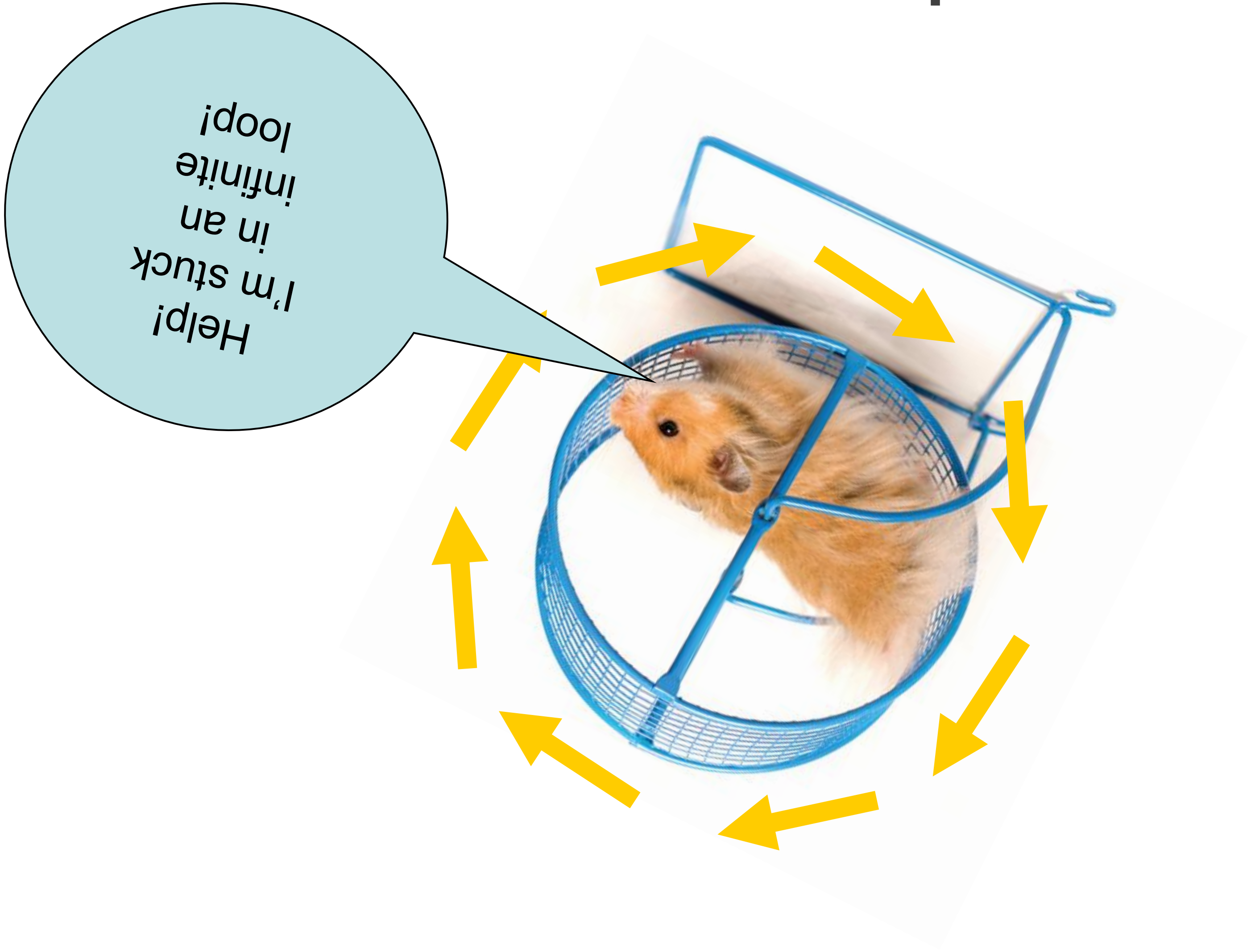
Common Error – Infinite Loops



Common Error – Infinite Loops



Common Error – Infinite Loops



Another infinite loop

while loop to hand-trace

```
1 i = 3;
2 while (i > 0);
  {
3     cout << i << " ";
4     i++;
  }
5 return 0
```

That semicolon causes the while loop to have an “empty body” which is executed forever.

trace

line	i	comment
1	3	
2	“	condition is true
2	“	condition is true
2	“	condition is true
2	“	condition is true
2	“	condition is true
2	“	condition is true
2	“	condition is true
2	“	condition is true
2	“	condition is true
2	“	condition is true
2	“	condition is true
2	“	condition is true
....		

Yet another infinite loop

while loop to hand-trace

```
1 i = 3;
2 while (i > 0)
  {
3    cout << i << " ";
4
  }
5 return 0
```

Forgetting to update the variable used in the condition is common.

trace

line	i	comment
1	3	
2	“	condition is true
3	3	print “3 “
2	“	condition is true
3	3	print “3 “
2	“	condition is true
3	3	print “3 “
2	“	condition is true
3	3	print “3 “
2	“	condition is true
3	3	print “3 “
2	“	condition is true
3	3	print “3 “
....		

Common Error – Infinite Loops

- In the investment program, forgetting to update might look like this.

```
year = 1;
while (year <= 20)
{
   balance = balance * (1 + RATE / 100);
}
```

- The variable **`year`** is not updated in the body

Common Error – Are We There Yet?

Well, are we?

Common Error – Are We There Yet?

When doing something repetitive,
most of us want to know when we are done.

For example, you may think,
"I want to get at least $20,000,"
and set the loop condition to

```
while (balance >= TARGET)
```

$$\overset{\uparrow}{\text{wrong test}}$$

Common Error – Are We There Yet?

But the **`while`** loop thinks the opposite:

How long am I allowed to keep going?

What is the correct loop condition?

```
while (                         )
```

Common Error – Are We There Yet?

But the **`while`** loop thinks the opposite:

How long am I allowed to keep going?

What is the correct loop condition?

```
while (balance < TARGET)
```

In other words:
"Keep at it while the balance
is less than the target".

Common Error – Are We There Yet?

When writing a loop condition, don’t ask, “Are we there yet?”

The *condition* determines how long the loop will keep going.

When writing a loop condition, ask, “Should we go on?”

Problem Solving: Hand-Tracing

Hand-tracing is a method of checking your work.

To do a hand-trace, write your variables on a sheet of paper and mentally execute each step of your code…

writing down the values of the variables
as they are changed in the code.

Keep track where you are in the program, and what values the variables have –

that way you can also see the history of the values.

Problem Solving

- Consider this example.

```
1  int n = 738;
2  int sum = 0;
3  int digit;
4  while(n>0){
5       digit=n % 10;
6       sum = sum + digit;
7       n = n/10;
8  }
9  cout << "Sum: " << sum;
```

- What is the hand trace?
- What is computed?

Line	n	sum	digit	comment
1	738			
2	"	0		
3	"	"		
4	"	"		condition true