Imperative programming
Loops

Lecture 04.01

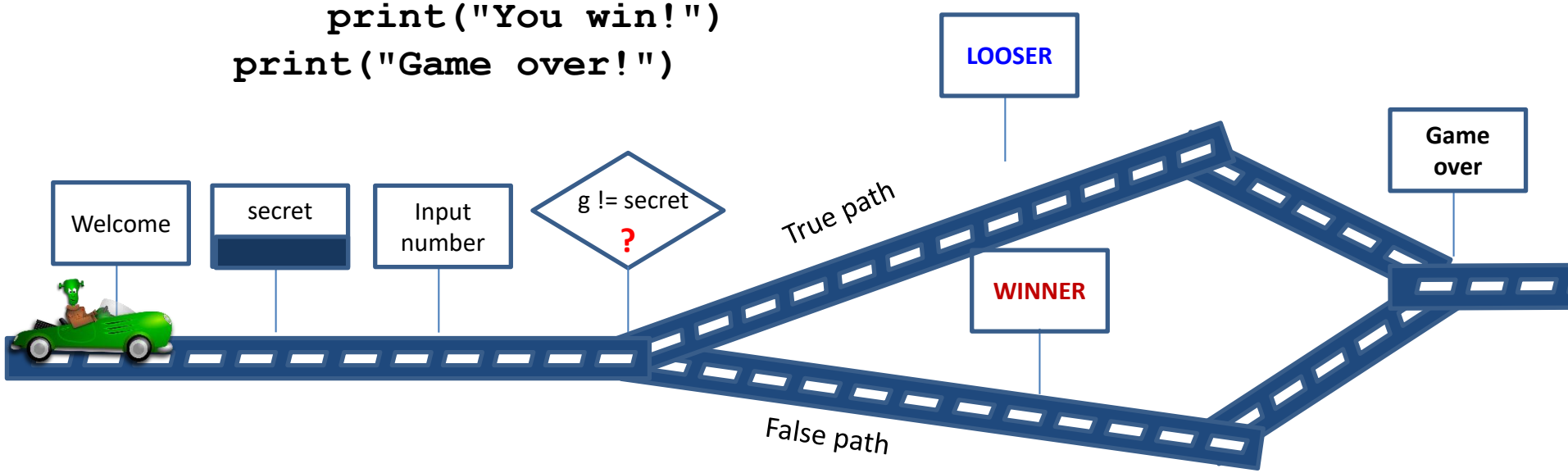
By Marina Barsky

Imperative programming!

- A programming paradigm that describes computation in terms of *statements* that change program *state* (change the values of common variables)
- The statements are performed one after another, according to the program path

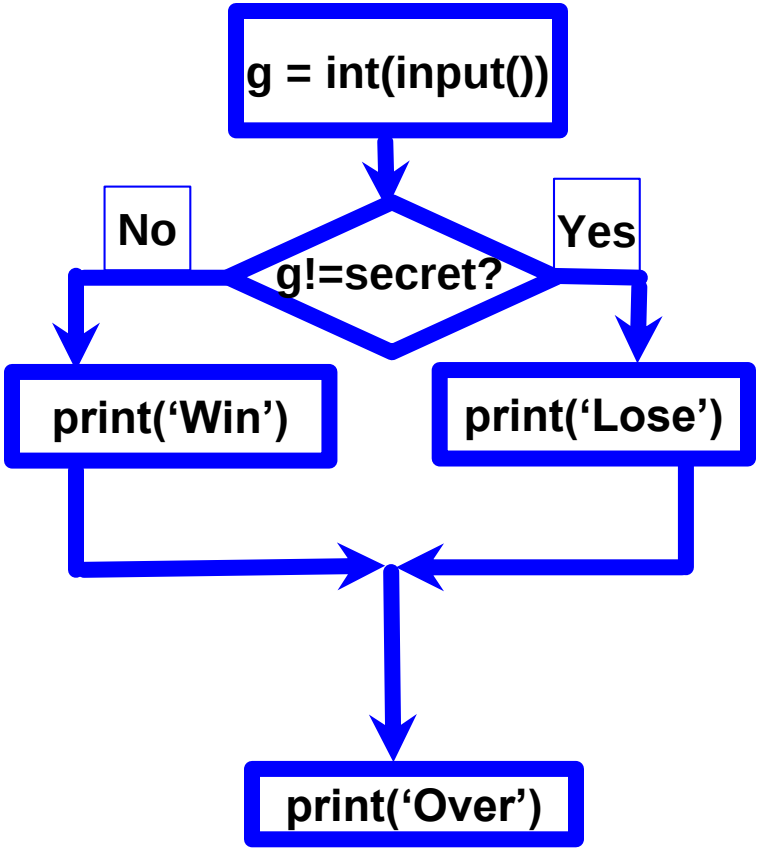
Guessing game

```
print("Welcome!")
secret = random.choice(range(10))
g = int(input
        ("Guess the number: "))
if g != secret:
    print("You lose!")
else:
    print("You win!")
print("Game over!")
```



The program as a network of branching roads

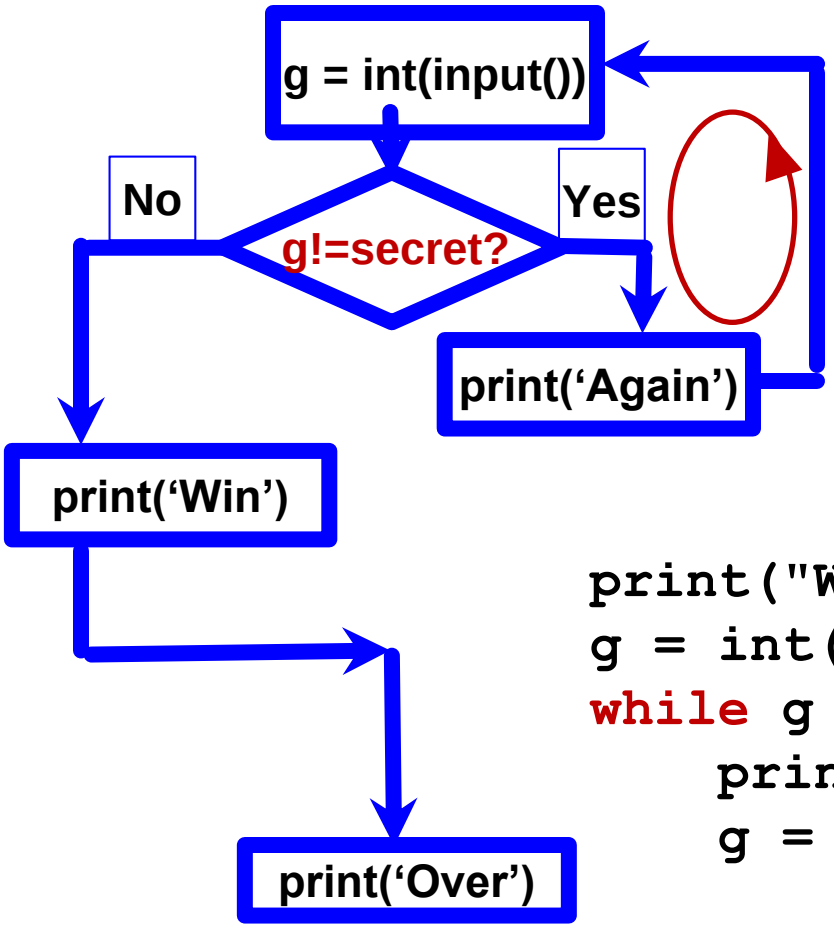
If/else branches




```
print("Welcome!")
secret = random.choice(range(10))
g = int(input
      ("Guess the number: "))
if g != secret:
    print("You lose!")
else:
    print("You win!")
print("Game over!")
```

We ask the question and change the program path according to the answer

Guessing game: giving another chance



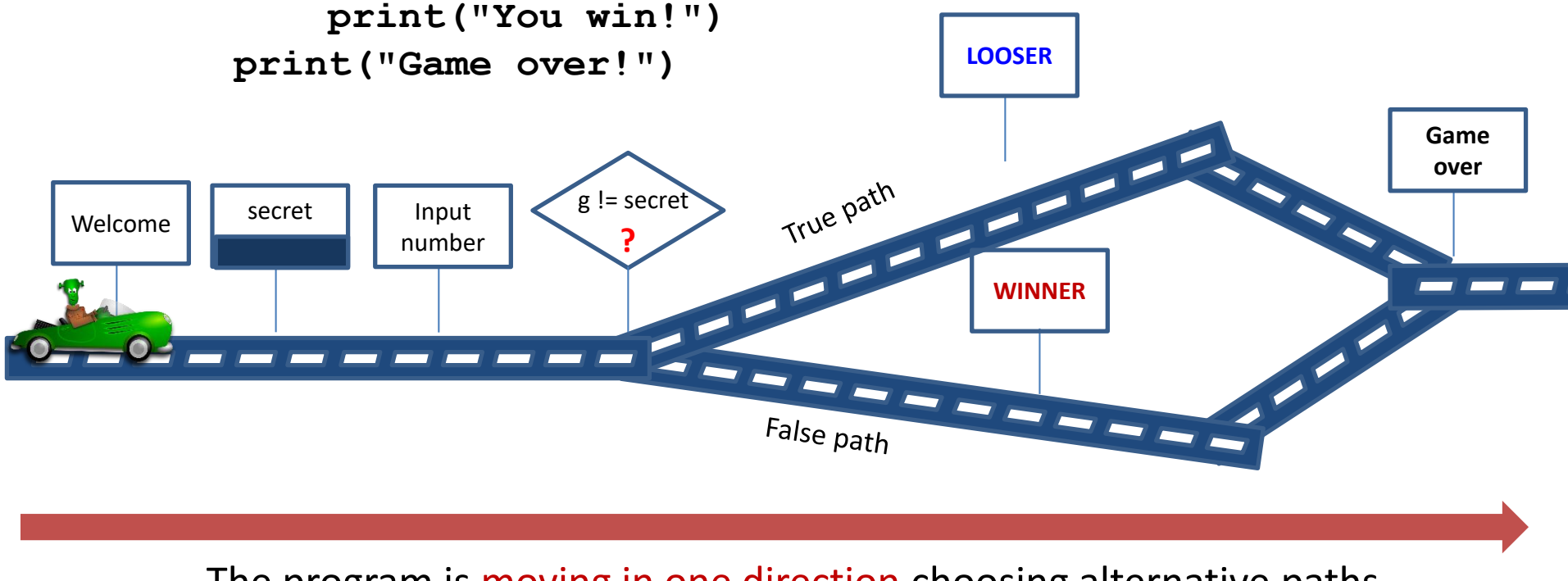
Loops allow you to repeat the sequence of commands **while a certain condition remains True**

```
print("Welcome!")
g = int(input("Guess the number: "))
while g != secret:
    print("Try again!")
    g = int(input ("Guess the number:"))

print("You win!")
print("Game over!")
```

Guessing game

```
print("Welcome!")
secret = random.choice(range(10))
g = int(input
        ("Guess the number: "))
if g != secret:
    print("You lose!")
else:
    print("You win!")
print("Game over!")
```

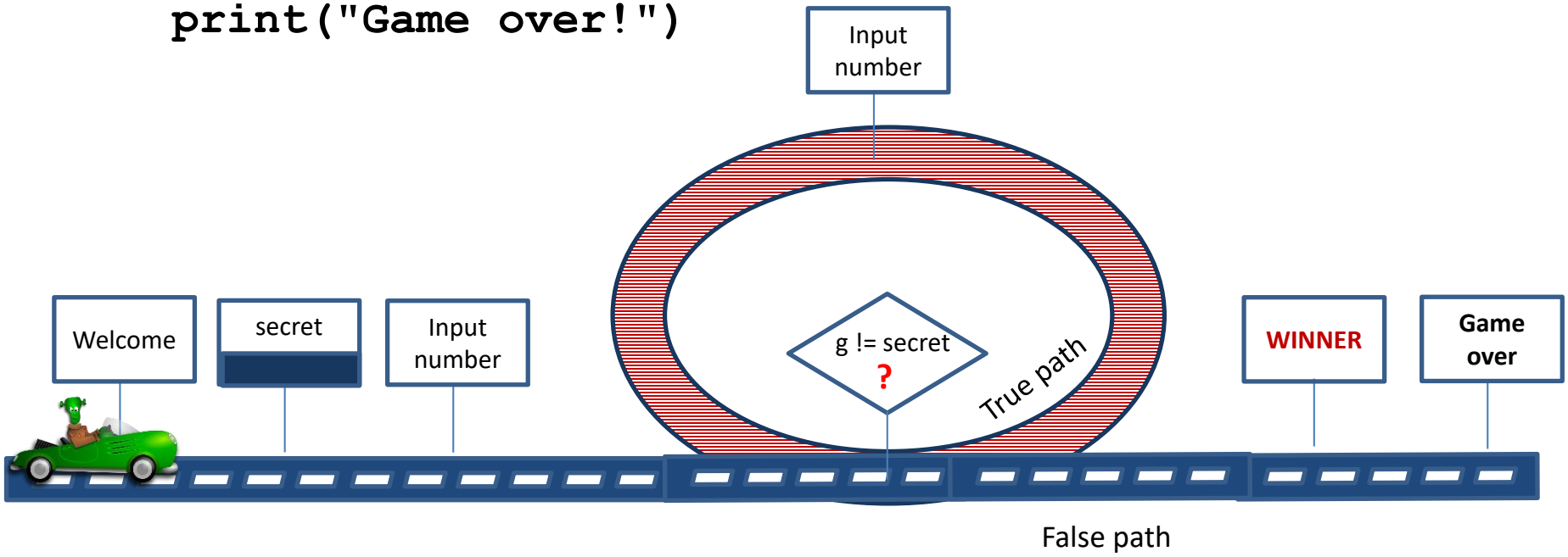


The program is moving in one direction choosing alternative paths

Guessing game

```
print("Welcome!")
g = int(input("Guess the number: "))
while g != secret:
    print("Try again!")
    g = int(input ("Guess the number:"))

print("You win!")
print("Game over!")
```



Roundabout

Another example

The user types in a sequence of positive integers. The program is to add up the integers.
The user signals the end of the input by entering a non-positive value which should not be part of the sum.

```
sum = 0
v = int(input('Enter a positive integer: '))
while ( v > 0):
    sum = sum + v
    v = int(input('Enter a positive integer: '))
print('Summation is :', sum )
```

Conditional **execution**

```
if <condition>:
   <body>
```

```
x = 5
if x != 0:
   print(x)
   x = x - 1
print(x)
```

What is printed on the screen?

Conditional **iteration** (repetition)

while <condition>:
  <loop body>

`while` loops:
indefinite, *conditional* iteration

```
x = 5
while x != 0:
   print(x)
   x = x - 1
print(x)
```

What is printed now?

The main pattern of `while` loop

Prime the condition.

while <condition>:
  <body>
  <update condition>

Check the condition.

Update the condition.

```
x = 10
while x != 0:
   print(x)
   x = x - 1
print(x)
```

From the logical point of view, `while` loop has three components:

1. Prime the condition
2. Check the condition
3. Update the condition

The conditional variable **has to change** in the loop body!

```
jar_empty = False
while not jar_empty:
   scoop()
wash_jar()
```

The conditional variable **has to change** in the loop body!

```
n = 0
while n < 10:
   lather()
   rinse()
dry_off()
```

Extreme Looping

What does this code do?

```
print('It keeps on')

while True:
    print('going and')


print('Phew! I\'m done!')
```

Extreme Looping

What does this code do?

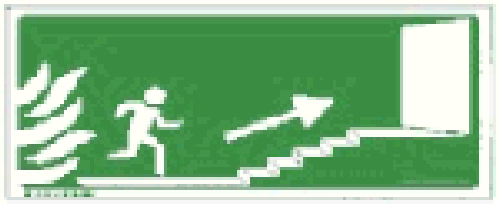
```
print('It keeps on')

while True:
    print('going and')


print('Phew! I\'m done!')
```

the loop keeps on running as long as this test is `True`

This won't print until the while loop finishes -
In this case, it ***never*** prints!



Making our escape!

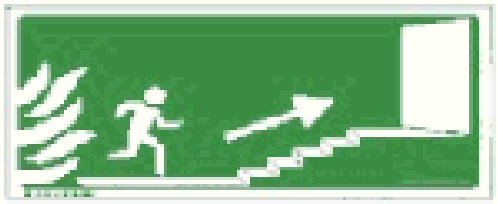
```
import random
escape = 0

while escape != 42:

    print('Help! Let me out!')
    escape = random.choice([41,42,43])


print('At last!')
```

Will the same thing appear every time this is run?



Making our escape!

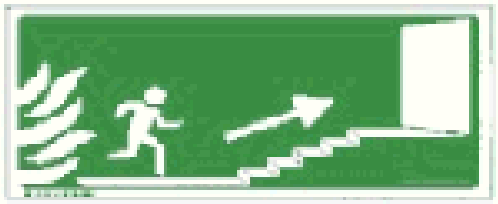
```
import random
escape = 0

while escape != 42:

    print('Help! Let me out!')
    escape = random.choice([41,42,43])


print('At last!')
```

What modifications do we need to make to count the number of iterations and report it?



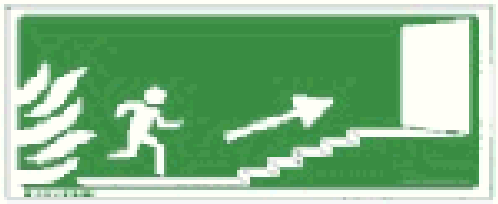
Count the iterations…

```
import random
escape = 0
count = 0

while escape != 42:
    count += 1
    print('Help! Let me out!')
    escape = random.choice([41,42,43])


print('At last!')
```

How could we make it harder/easier to escape?



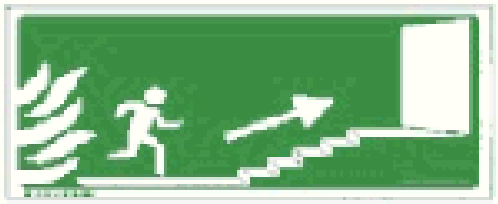
Harder to escape

```
import random
escape = 0
count = 0

while escape != 42:
    count += 1
    print('Help! Let me out!')
    escape = random.choice(range(100))


print('At last!')
```

How could we make the escape after at most 10 iterations?



Ensuring the escape

```
import random
escape = 0
count = 0

while escape != 42:
    count += 1
    print('Help! Let me out!')
    escape = random.choice(range(100))
    if count == 10:
        break

print('At last!')
```

break and continue

```
while True:
    line = input('> ')
    if line[0] == '#' :
        continue
    if line == 'done' :
        break
    print(line)
print ('goodbye')
```

Continue to the next iteration of the loop

Break out of the loop and do the next command after loop body

while we're here...

```
x = 39
while x < 44:
    if x % 2 == 0:
        print(x)
    x += 1
```

What numbers will this loop print out?

while

is commonly used for input validation

```
a = int(input("Number between 1 and 10: "))
while not (a >= 1 and a <= 10):
    a = int(input("Number between 1 and 10! "))
print("Finally!")
```

This will not let users out until they enter the required number

`while` loop summary

- Condition variable should be initialized (outside the loop) in order to enter the loop
- Condition should be updated in the body of the loop
- If you need to accumulate a value during loop iterations, the accumulator variable should be initialized *outside the loop*

Exercise

- Ask the user to enter a series of numbers and when the user enters ‘#’ report an average.