CS101 - Introduction to Programming

Lecture 1

School of Computing
KAIST

Roadmap

- Introduction to CS101
- Introduction to Programming

Welcome to CS101!

CS101

- 11 lectures, 10 lab sections; 5 lecturers
- 6 lead TAs; 36 helper TAs

Welcome to CS101!

CS101

- 11 lectures, 10 lab sections; 5 lecturers
- 6 lead TAs; 36 helper TAs

All sections have one 2-hour lecture per week by professors (Tuesday 10:30AM ~12:30PM).
Each section has one 3-hour lab per week led by TAs.
This is **the most important part of the course**!

- Sections A and B (Moonzoo Kim): Monday (9am-12pm)
- Sections C and D (Junehwa Song): Tuesday (1pm-4pm)
- Sections E and F (Geehyuk Lee): Wednesday (9am-12pm)
- Sections G and H (Soontae Kim): Thursday (1pm-4pm)
- Sections I and J (Young Hee Lee): Friday (9am-12pm)

Grading

Practice points:

- 100 points for lecture attendance
- 100 points for lab work
- 150 points for homework: 50 points per assignment
- 0 point for extra class

Students need to collect at least 300 practice points.

Grading

Practice points:

- 100 points for lecture attendance
- 100 points for lab work
- 150 points for homework: 50 points per assignment
- 0 point for extra class

Students need to collect at least 300 practice points.

Theory points:

- 100 points for midterm exam
- 100 points for final exam.

The grade is determined **by the theory points only**. The practice points over 300 are ONLY qualification for grading.

Section Change/New Registration

No section change is allowed

New Registration:

- You can apply for CS101 course only on the web (Portal $\rightarrow$ Academic system) during the add/drop period
- Those whose majors are EE, SoC, ICE, or ISE dept. and who should re-take CS101 should contact InYang Hwang (inyang@kaist.ac.kr)

Course Drop:

- Contact InYang Hwang (inyang@kaist.ac.kr) @ School of Computing office located at E3 1st floor

We do not accept add/drop form in paper/person

Lecture Attendance

No attendance check in the first lecture.

When you come to the 2nd lecture

- Pick a seat and it will be your seat for the rest of the semester

From the 2nd to 11th lecture

- TA takes photos to check your attendance until 10:50AM
- Only students who are taken the photos get the attendance points
- No excuse or exception on the lecture attendance

Honesty policy

Cheating is strongly forbidden.

Cheating on homework or exams will give an F.

We have one TA dedicated to checking plagiarism.

- In Spring 2015, 21 got an F for cheating.
- In Fall 2015, 2 got an F for cheating.
- In Spring 2016, 7 got an F for cheating.
- In Fall 2016, 7 got an F for cheating.
- In Spring 2017, 7 got an F for cheating.
- In Fall 2017, 2 got an F for cheating.

Don’t make your friend get an F by copying.

Furthermore, KAIST student examination committee will officially punish those who deny their cheating activities (i.e., suspension, expulsion, 정학, 퇴학)

Course material

All course related materials will be made available on the course website.

- Lecture slides and sample codes
- Lecture notes on robot programming and photo processing
- Lab materials
- http://cs101.kaist.ac.kr

Main reference: Python for software design by Allen B. Downey, Cambridge, 2009.

Python

This course is about programming and computational thinking, not about learning a programming language.

Python

This course is about programming and computational thinking, not about learning a programming language.

Python is a programming language that is easy to learn and very powerful.

- Used in many universities for introductory courses.
- Main language used for web programming at Google.
- Widely used in scientific computation, for instance at NASA, by mathematicians and physicists.
- Available on embedded platforms, for instance Nokia mobile phones.
- Large portions of games (such as Civilization IV) are written in Python.

Python

This course is about programming and computational thinking, not about learning a programming language.

Python is a programming language that is easy to learn and very powerful.

- Used in many universities for introductory courses.
- Main language used for web programming at Google.
- Widely used in scientific computation, for instance at NASA, by mathematicians and physicists.
- Available on embedded platforms, for instance Nokia mobile phones.
- Large portions of games (such as Civilization IV) are written in Python.

Once you learnt programming in one language, it is relatively easy to learn another language, such as C++ or Java.

Why are you here?

Every scientist and engineer must know some programming. It is part of basic education, like calculus, linear algebra, introductory physics and chemistry, or English.

Alan Perlis 1961

Why are you here?

Every scientist and engineer must know some programming. It is part of basic education, like calculus, linear algebra, introductory physics and chemistry, or English.

Alan Perlis 1961

Computer science is not computer programming. We teach programming to teach **computational thinking**:

- Solving problems (with a computer).
- Thinking on multiple levels of abstraction.
  Decompose a problem into smaller problems.
- A way of human thinking (**not** “thinking like a computer")
- Thinking about recipes (**algorithms**).
  30 years ago the solution to a problem in science or engineering was usually a formula. Today it is usually an algorithm (DNA, proteins, chemical reactions, factory planning, logistics).

What is a program?

A program is a **sequence of instructions** that solves some problem (or achieves some effect).

What is a program?

A program is a **sequence of instructions** that solves some problem (or achieves some effect).

For instance: Call your friend on the phone and give her instructions to find your favorite cafe. Or explain how to bake a cake.

What is a program?

A program is a **sequence of instructions** that solves some problem (or achieves some effect).

For instance: Call your friend on the phone and give her instructions to find your favorite cafe. Or explain how to bake a cake.

Instructions are operations that the computer can already perform.

What is a program?

A program is a **sequence of instructions** that solves some problem (or achieves some effect).

For instance: Call your friend on the phone and give her instructions to find your favorite cafe. Or explain how to bake a cake.

Instructions are operations that the computer can already perform.

But we can define **new instructions** and raise the **level of abstraction**!

What is a program?

A program is a **sequence of instructions** that solves some problem (or achieves some effect).

For instance: Call your friend on the phone and give her instructions to find your favorite cafe. Or explain how to bake a cake.

Instructions are operations that the computer can already perform.

But we can define **new instructions** and raise the **level of abstraction**!

A program implements an **algorithm** (a recipe for solving a problem).

What is debugging?

A **bug** is a mistake in a program. **Debugging** means to find the mistake and to fix it.

What is debugging?

A **bug** is a mistake in a program. **Debugging** means to find the mistake and to fix it.

Computer programs are very complex systems. Debugging is similar to an experimental science: You experiment, form hypotheses, and verify them by modifying your program.

What is debugging?

A **bug** is a mistake in a program. **Debugging** means to find the mistake and to fix it.

Computer programs are very complex systems. Debugging is similar to an experimental science: You experiment, form hypotheses, and verify them by modifying your program.

Kinds of errors:

- **Syntax error**. Python cannot understand your program, and refuses to execute it.
- **Runtime error**. When executing your program (**at runtime**), your program suddenly terminates with an error message.
- **Semantic error**. Your program runs without error messages, but does not do what it is supposed to do.

Why is programming useful?

- 20 years ago, **electrical engineering students** learnt about circuits. Today they learn about embedded systems.
- You can build **a radio** in software.
- **Industrial engineers** program industrial robots. Moreover, today's industrial engineers work on logistics|problems that can only be solved by computer.
- **Modern automobiles** contain thousands of lines of code, and would not run without microprocessors.
- **Mathematicians** gain insight and intuition by experimenting with mathematical structures, even for discrete objects such as groups and graphs.
- **Experimental data** often needs to be reformatted to be analyzed or reused in different software. Python is fantastic for this purpose.
- **Experimental data sets** are nowadays too large to be handled manually.

Why learn programming?

- If you can only use software that **someone else** made for you, you limit your ability to achieve what you want.
- For instance, digital media is manipulated by software. If you can only use Photoshop, you limit your ability to express yourself.
- Programming gives you freedom.

Why is programming fun?

Programming is a creative process.

Why is programming fun?

Programming is a creative process.

A single person can actually build a software system of the complexity of the space shuttle hardware. Nothing similar is true in any other discipline.

Why is programming fun?

Programming is a creative process.

A single person can actually build a software system of the complexity of the space shuttle hardware. Nothing similar is true in any other discipline.

There is a large and active **open-source community**: people who write software in their free time for fun, and distribute it for free on the internet. For virtually any application there is code available that you can download and modify freely.

Let's get started

But now let me show you some Python code . . .

- Interactive Python
- Python programs (scripts)
- Comments
- Your own instructions: functions
- Keywords
- for loops
- Indentation

Adding new functions

A **function definition** specifies the name of a new function and the sequence of statements that execute when the function is **called**.

Adding new functions

A **function definition** specifies the name of a new function and the sequence of statements that execute when the function is **called**.

```
def print_message():
  print("CS101 is fantastic!")
  print("Programming is fun!")
```

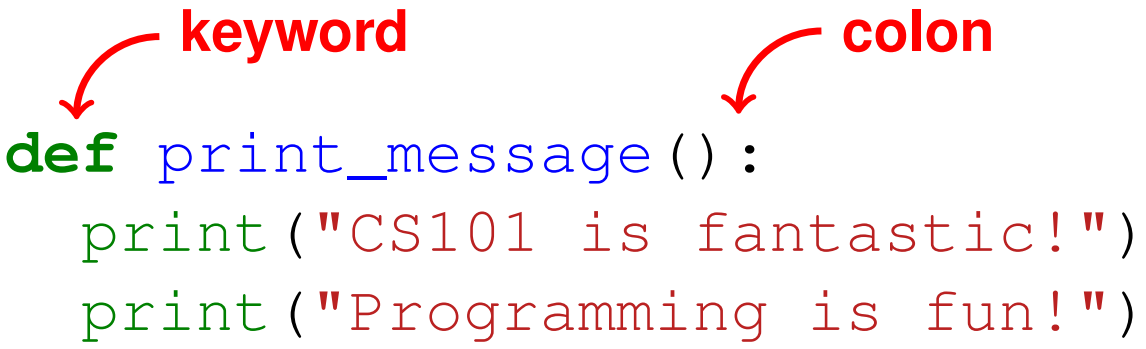
Adding new functions

A **function definition** specifies the name of a new function and the sequence of statements that execute when the function is **called**.


```
def print_message():
    print("CS101 is fantastic!")
    print("Programming is fun!")
```

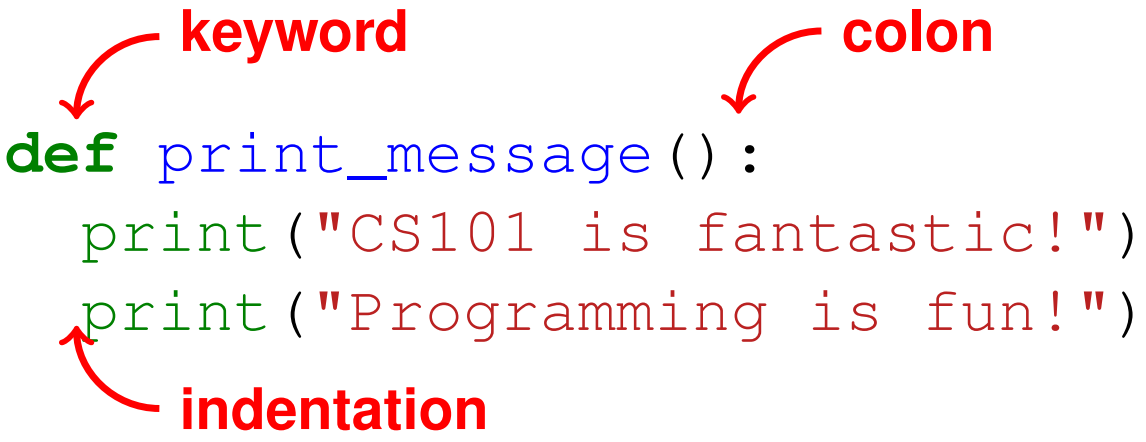
Adding new functions

A **function definition** specifies the name of a new function and the sequence of statements that execute when the function is **called**.



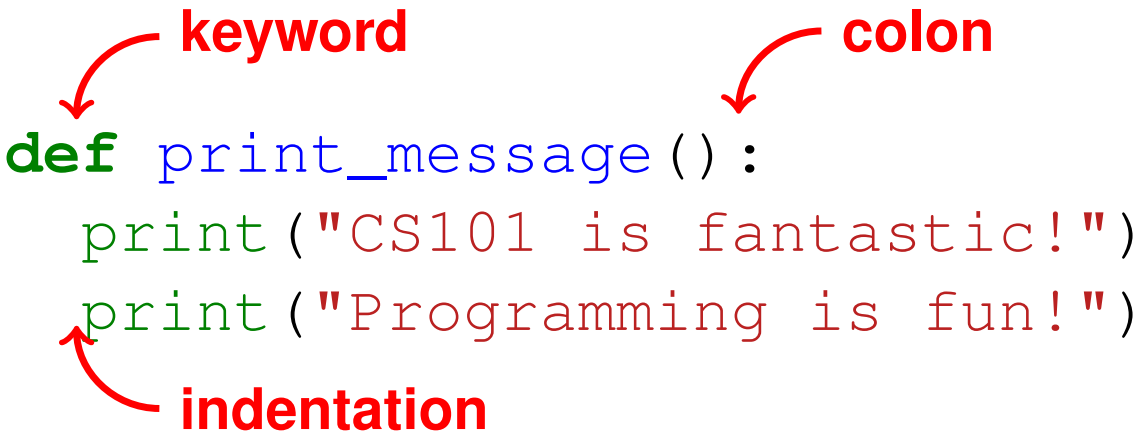
Adding new functions

A **function definition** specifies the name of a new function and the sequence of statements that execute when the function is **called**.



Adding new functions

A **function definition** specifies the name of a new function and the sequence of statements that execute when the function is **called**.



You can call a function inside another function:

```
def repeat_message():
    print_message()
    print_message()
```

Flow of execution

```
def print_message():
  print("CS101 is fantastic!")
  print("Programming is so much fun!")

def repeat_message():
  print_message()
  print_message()

repeat_message()
```

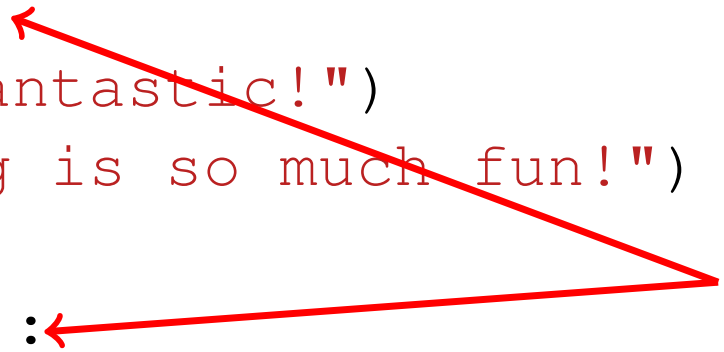
Flow of execution

```
def print_message():
  print("CS101 is fantastic!")
  print("Programming is so much fun!")

def repeat_message():
  print_message()
  print_message()

repeat_message()
```

Execution begins at the first statement. Statements are executed one-by-one, top to bottom.

Flow of execution

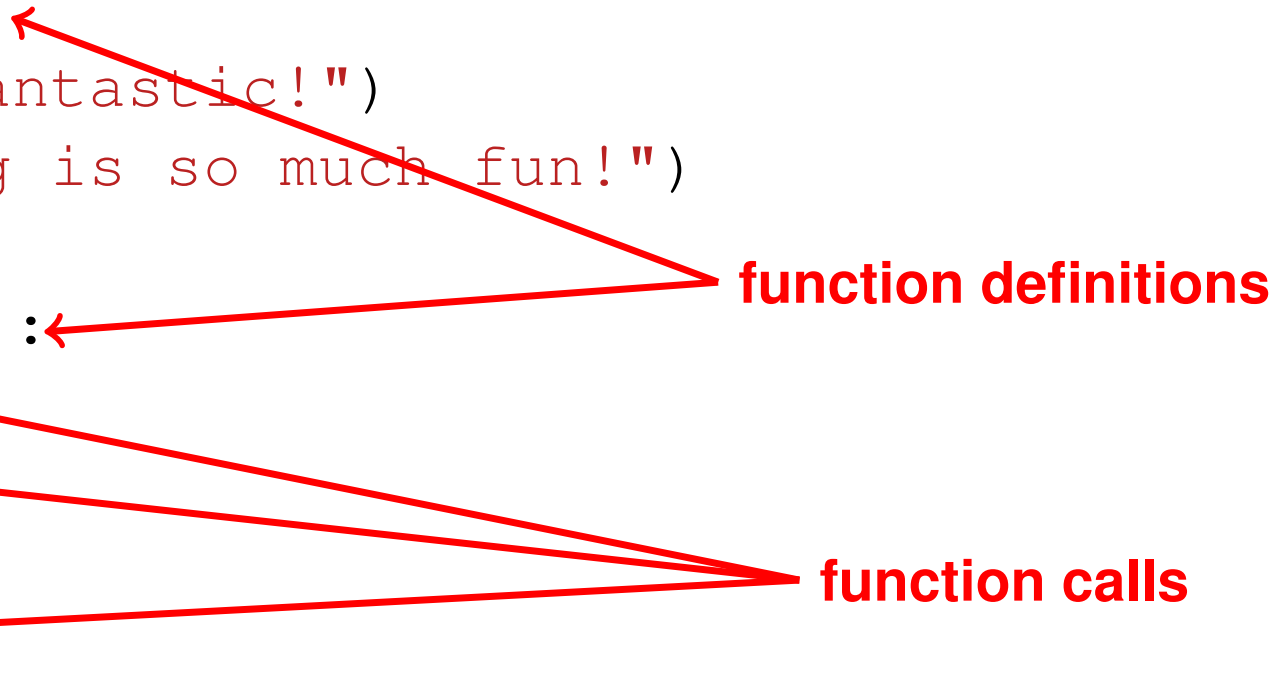
```
def print_message():
    print("CS101 is fantastic!")
    print("Programming is so much fun!")

def repeat_message():
    print_message()
    print_message()

repeat_message()
```



Execution begins at the first statement. Statements are executed one-by-one, top to bottom.
Function **definitions** do not change the flow of execution But only **define** a function.

Flow of execution

```
def print_message():
    print("CS101 is fantastic!")
    print("Programming is so much fun!")

def repeat_message():
    print_message()
    print_message()

repeat_message()
```



Execution begins at the first statement. Statements are executed one-by-one, top to bottom.

Function **definitions** do not change the flow of execution But only **define** a function.

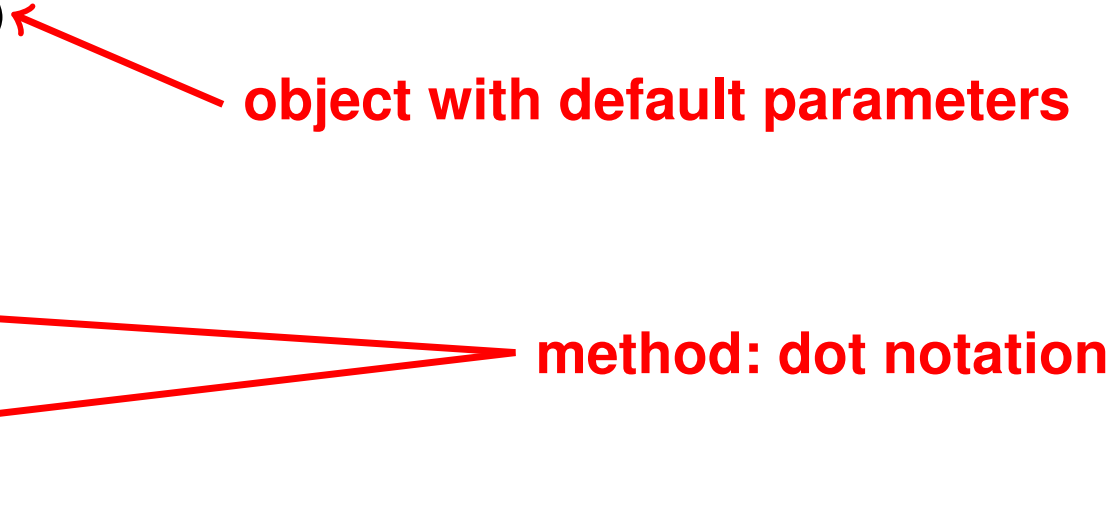
Function **calls** are like **detours** in the flow of execution.

Robots

```
# create a robot with one beeper
hubo = Robot(beepers = 1)

# move one step forward
hubo.move()

# turn left 90 degrees
hubo.turn_left()
```

Robots

```
# create a robot with one beeper
hubo = Robot(beepers = 1)

# move one step forward
hubo.move()

# turn left 90 degrees
hubo.turn_left()
```



Robots

```
# create a robot with one beeper
hubo = Robot(beepers = 1)

# move one step forward
hubo.move()

# turn left 90 degrees
hubo.turn_left()
```

object with default parameters

method: dot notation

How can `hubo` turn right?

Robots

```
# create a robot with one beeper
hubo = Robot(beepers = 1)

# move one step forward
hubo.move()

# turn left 90 degrees
hubo.turn_left()
```

object with default parameters

method: dot notation

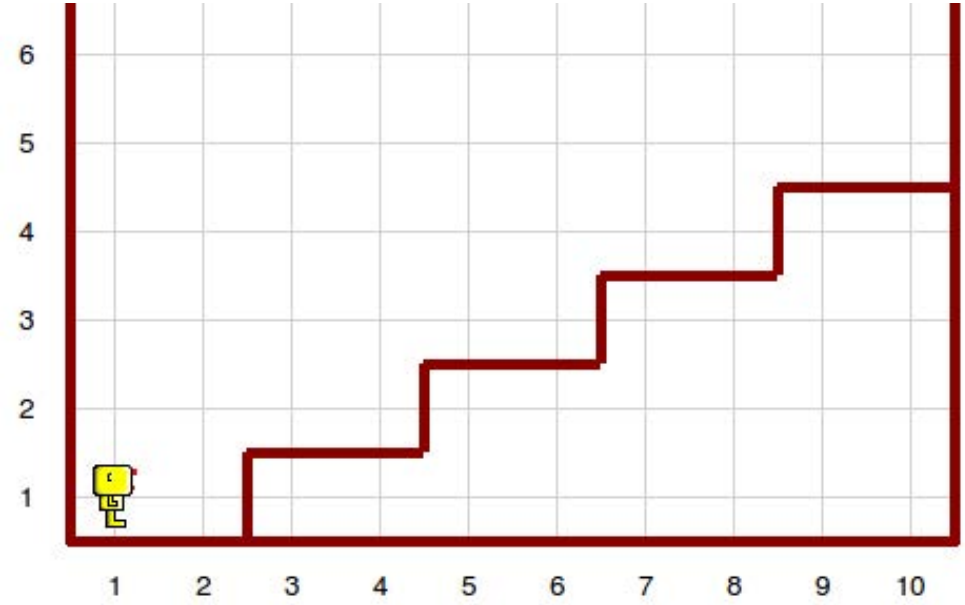
How can `hubo` turn right?

Define a function!

```
def turn_right():
    hubo.turn_left()
    hubo.turn_left()
    hubo.turn_left()
```

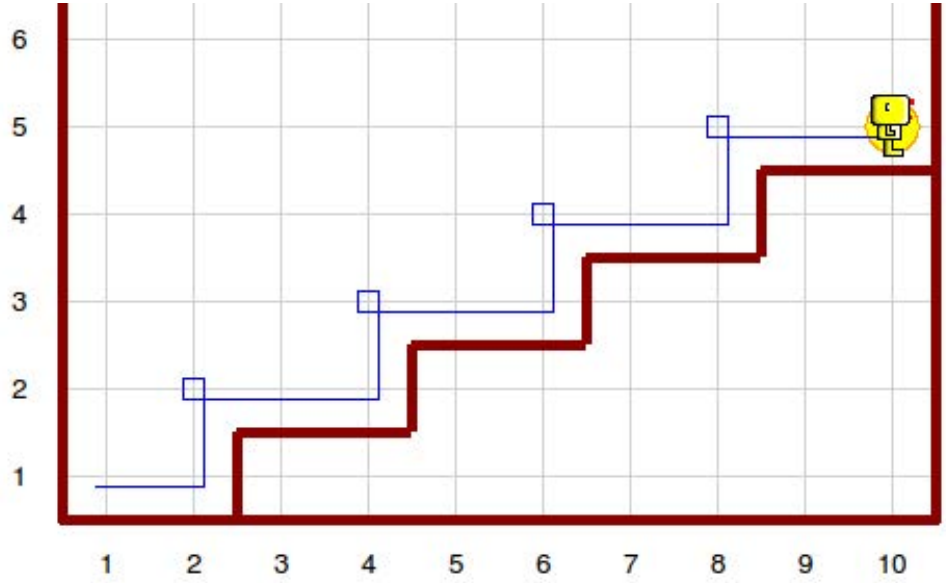
Newspaper delivery

Hubo should climb the stairs to the front door, drop a newspaper there, and return to his starting point.



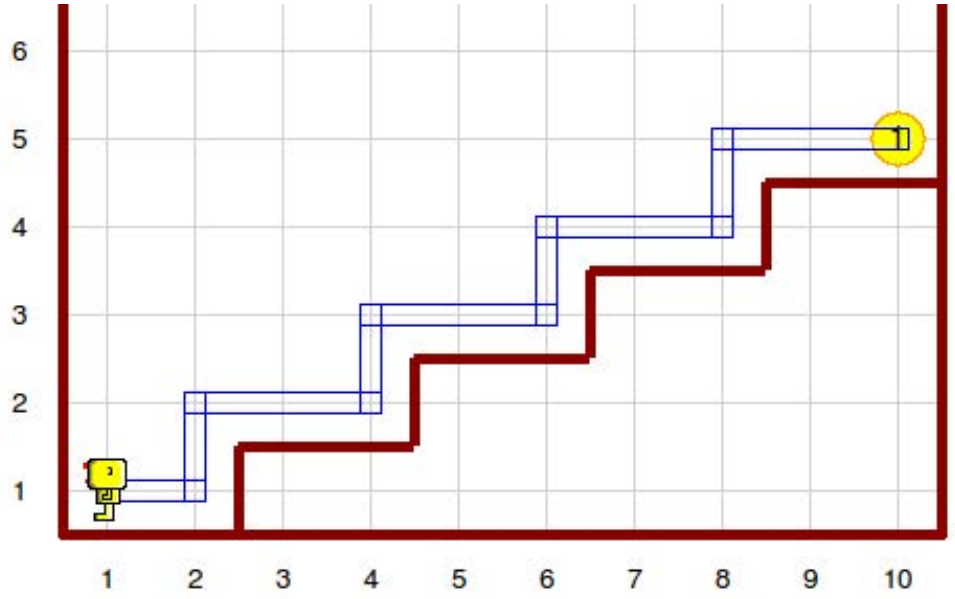
Newspaper delivery

Hubo should climb the stairs to the front door, drop a newspaper there, and return to his starting point.



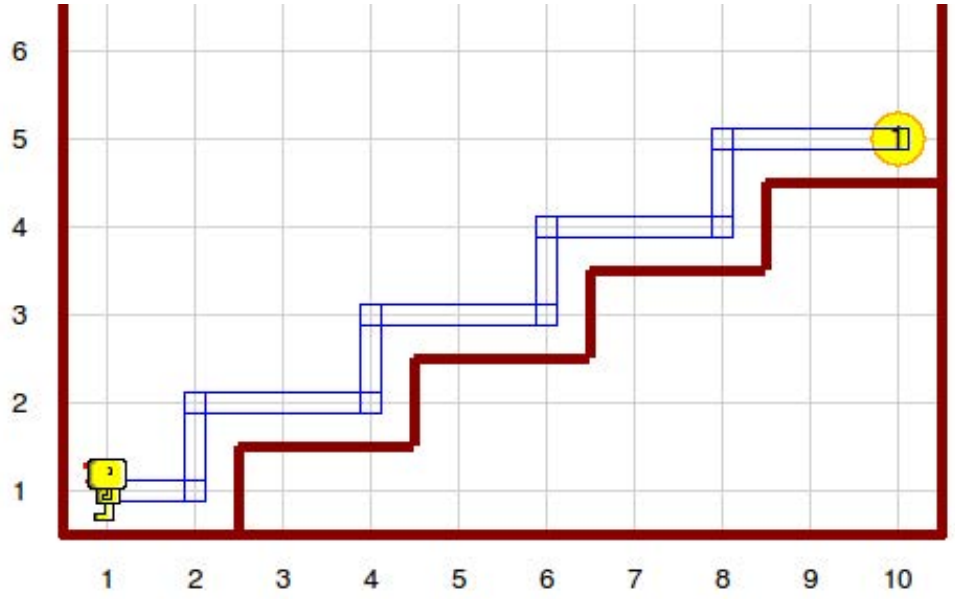
Newspaper delivery

Hubo should climb the stairs to the front door, drop a newspaper there, and return to his starting point.



Newspaper delivery

Hubo should climb the stairs to the front door, drop a newspaper there, and return to his starting point.

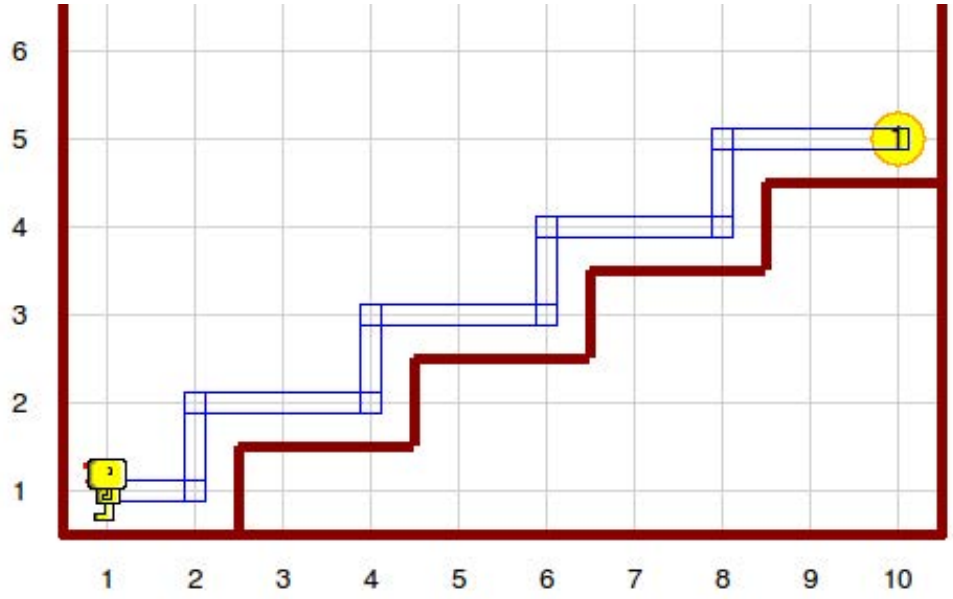


Problem outline:

- Climb up four stairs
- Drop the newspaper
- Turn around
- Climb down four stairs

Newspaper delivery

Hubo should climb the stairs to the front door, drop a newspaper there, and return to his starting point.



Problem outline:

- Climb up four stairs
- Drop the newspaper
- Turn around
- Climb down four stairs

Python version:

```
climb_up_four_stairs()
hubo.drop_beeper()
turn_around()
climb_down_four_stairs()
```

Newspaper delivery

```
def turn_around():
    hubo.turn_left()
    hubo.turn_left()
```

Newspaper delivery

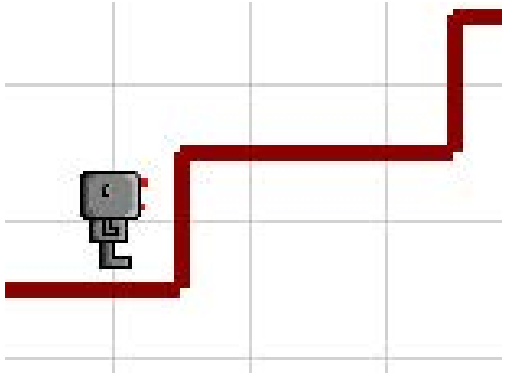
```
def turn_around():
    hubo.turn_left()
    hubo.turn_left()

def climb_up_four_stairs():
    # how?
```

Newspaper delivery

```
def turn_around():
  hubo.turn_left()
  hubo.turn_left()

def climb_up_four_stairs():
  climb_up_one_stair()
  climb_up_one_stair()
  climb_up_one_stair()
  climb_up_one_stair()
```

Newspaper delivery

```
def turn_around():
    hubo.turn_left()
    hubo.turn_left()

def climb_up_four_stairs():
    climb_up_one_stair()
    climb_up_one_stair()
    climb_up_one_stair()
    climb_up_one_stair()

def climb_up_one_stair():
    hubo.turn_left()
    hubo.move()
    turn_right()
    hubo.move()
    hubo.move()
```



Top-down design

Start at the top of the problem, and make an outline of a solution.

Top-down design

Start at the top of the problem, and make an outline of a solution.

For each step of this solution, either write code directly, or outline a solution for this step.

Top-down design

Start at the top of the problem, and make an outline of a solution.

For each step of this solution, either write code directly, or outline a solution for this step.

When all the partial problems have become so small that we can solve them directly, we are done and the program is finished.

Simple repetitions

To repeat the same instruction 4 times:

```
for i in range(4):
  print("CS101 is fantastic!")
```

Simple repetitions

To repeat the same instruction 4 times:

```
for i in range(4):
  print("CS101 is fantastic!")
```

Simple repetitions

To repeat the same instruction 4 times:

```
for i in range(4):
    print("CS101 is fantastic!")
```

for-loop

Don’t forget the indentation!

Simple repetitions

To repeat the same instruction 4 times:

```
for i in range(4):
  print("CS101 is fantastic!")
```

for-loop

Don't forget the indentation!

What is the difference:

```
for i in range(4):
  print("CS101 is great!")
  print("I love programming!")
```

and

```
for i in range(4):
  print("CS101 is great!")
print("I love programming!")
```

Avoiding repetitions

```
def climb_up_four_stairs():
    climb_up_one_stair()
    climb_up_one_stair()
    climb_up_one_stair()
    climb_up_one_stair()
```

Avoiding repetitions

```
def climb_up_four_stairs():
  climb_up_one_stair()
  climb_up_one_stair()
  climb_up_one_stair()
  climb_up_one_stair()
```

We should avoid writing the same code repeatedly. A **for**-loop allows us to write this more elegantly:

```
def climb_up_four_stairs():
  for i in range(4):
    climb_up_one_stair()
```