

Fall 2024

CS101 - Introduction to Programming

Lecture 1

School of Computing, KAIST

Welcome to CS101!

CS101

- 11 lectures, 10 lab sections; 2 lecturers
- 6 lead TAs; 40 helper TAs
- Homepage: <http://cs101.kaist.ac.kr/>

We have one **2-hour lecture** per week taught by professors (Monday 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, B, C, D, & E (Prof. Moonzoo Kim)
- Sections F, G, H, I, & J (Prof. Jongmoon Baik)

- Lectures

- Lectures will be given in person
- Lecture room:
 - E3-1. Rm. 1501 – A, B, C, D, E
 - Creative Bldg., Turman Hall – F, G, H, I, J
- Attendance check
 - Students access the QR of each seat and submit the form within 15 mins.
 - TA will check for empty seats at a random time after the start of class.
 - Attendance will be regarded as absent when,
 - students use the same QR code.
 - students use the QR code of empty seats.

- Lab Sessions

- All lab sessions will be held in person
- Lab. dates, times & lecture rooms:

	MON	TUE	WED	THU	FRI
9:00-12:00	A&B	C&D		G&H	I&J
13:00-16:00	A&B		E&F		

Section Change/New Registration

No section change is allowed.

New Registration:

You can apply for CS101 course only on the web (Portal → Academic system) during the add/drop period.

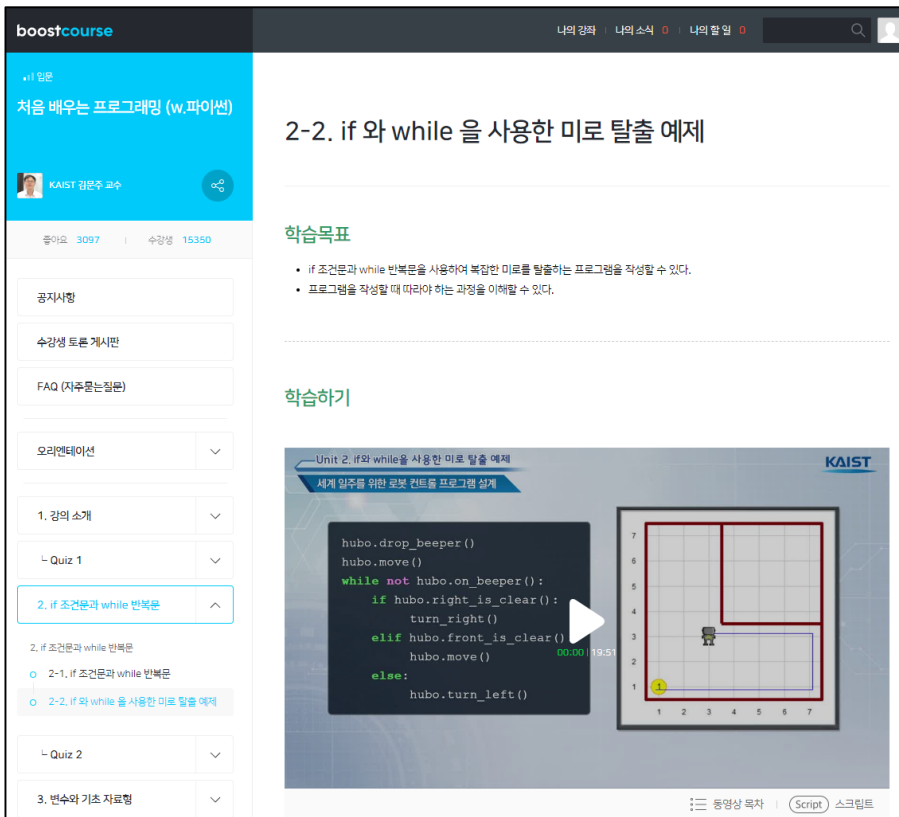
Those whose majors are EE, SoC, ICE, or ISE dept. and who should re-take CS101 should contact [Ms. Seong Hee Kang \(diane813@kaist.ac.kr\)](mailto:diane813@kaist.ac.kr).

Course Drop:

Contact [Ms. Seong Hee Kang \(diane813@kaist.ac.kr\)](mailto:diane813@kaist.ac.kr) @ KAIST
School of Computing office located at E3-1 1st floor Rm. 1402

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

- You can register and learn the on-line CS101 lectures in Naver Boostcourse 처음 배우는 프로그래밍 (w.파이썬)
 - <https://www.boostcourse.org/cs114/intro>
- The on-line CS101 lectures can help those who have difficulties in CS101
 - The on-line lectures provide detailed visual explanations
 - You can repeatedly watch videos until you understand
 - The on-line lectures are taught in Korean (한국어 강의)



boostcourse

나의 강좌 | 나의 소식 | 나의 할 일

처음 배우는 프로그래밍 (w.파이썬)

KAIST 김문주 교수

좋아요 3097 | 수강생 15350

공지사항

수강생 토론 게시판

FAQ (자주 묻는 질문)

오리엔테이션

1. 강의 소개

Quiz 1

2. if 조건문과 while 반복문

2. if 조건문과 while 반복문

2-1. if 조건문과 while 반복문

2-2. if와 while을 사용한 미로 탈출 예제

Quiz 2

3. 변수와 기초 자료형

2-2. if와 while을 사용한 미로 탈출 예제

학습목표

- if 조건문과 while 반복문을 사용하여 복잡한 미로를 탈출하는 프로그램을 작성할 수 있다.
- 프로그램을 작성할 때 따라야 하는 과정을 이해할 수 있다.

학습하기

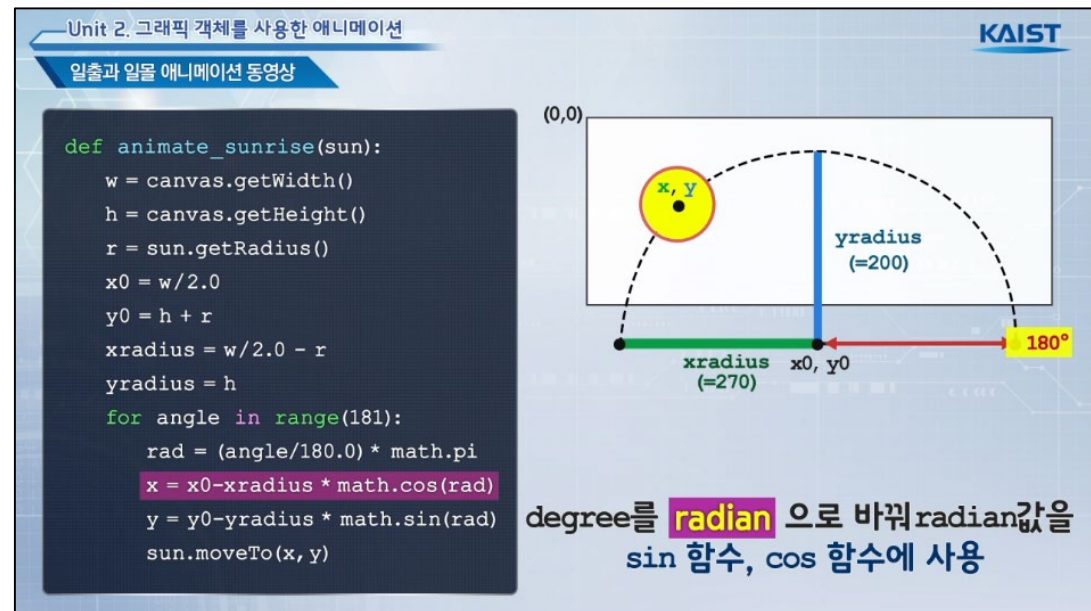
Unit 2. if와 while을 사용한 미로 탈출 예제

세계 일주를 위한 로봇 컨트롤 프로그램 설계

```
hubo.drop_beeper()
hubo.move()
while not hubo.on_beeper():
    if hubo.right_is_clear():
        turn_right()
    elif hubo.front_is_clear():
        hubo.move()
    else:
        hubo.turn_left()
```

KAIST

동영상 목차 | Script 스크립트



Unit 2. 그래픽 객체를 사용한 애니메이션

일출과 일몰 애니메이션 동영상

```
def animate_sunrise(sun):
    w = canvas.getWidth()
    h = canvas.getHeight()
    r = sun.getRadius()
    x0 = w/2.0
    y0 = h + r
    xradius = w/2.0 - r
    yradius = h

    for angle in range(181):
        rad = (angle/180.0) * math.pi
        x = x0 - xradius * math.cos(rad)
        y = y0 - yradius * math.sin(rad)
        sun.moveTo(x, y)
```

(0,0)

x, y

yradius (=200)

xradius (=270)

x0, y0

180°

degree를 **radian** 으로 바꿔radian값을 sin 함수, cos 함수에 사용

Practice points:

- 100 points for lecture attendance
- 100 points for lab work
- 100 points for homework

Students need to collect **at least 250 practice points**.

Theory points:

- Midterm exam – 40%
- Final exam – 60%

The final score for CS101 is determined **by the theory points only**.

You may get F if you failed to get enough theory points.

The practice points over 250 are ONLY qualification for grading.

Honesty Policy

Cheating is strongly forbidden

(If you look at even a single line of code written by someone else, it's cheating)

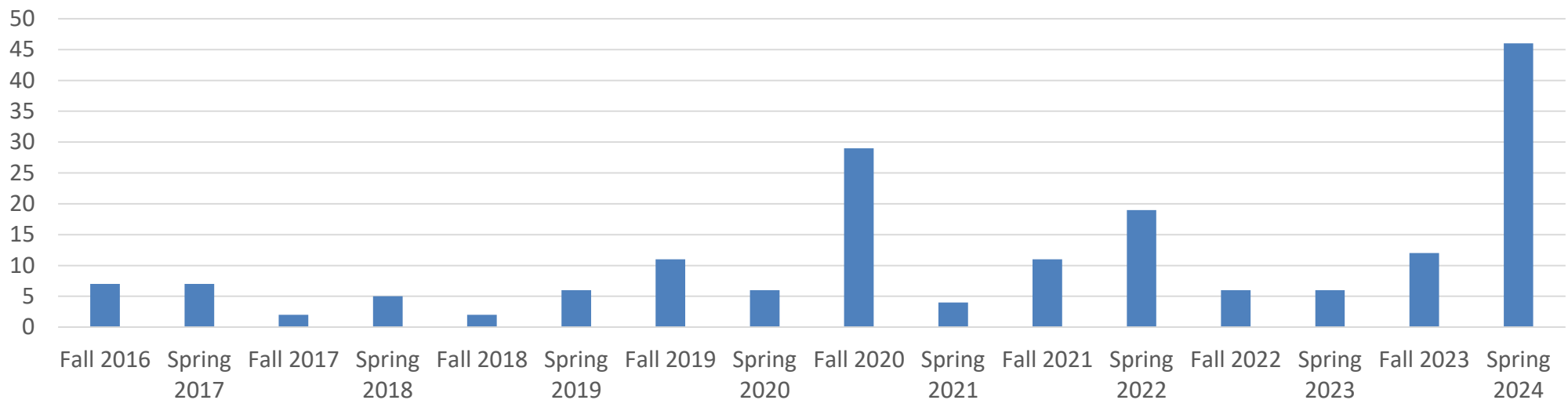
Cheating on homework or exams will give F.

We have a TA dedicated to checking plagiarism.

Don't make your friend get F by copying.

Fall 2016	Spring 2017	Fall 2017	Spring 2018	Fall 2018	Spring 2019	Fall 2019	Spring 2020	Fall 2020	Spring 2021	Fall 2021	Spring 2022	Fall 2022	Spring 2023	Fall 2023	Spring 2024
7	7	2	5	2	6	11	6	29	4	11	19	6	6	12	46

Number of students who got F for cheating



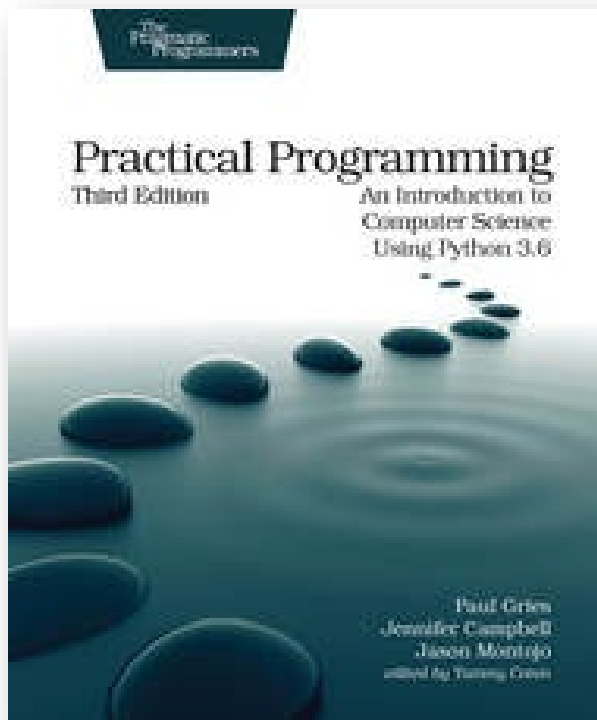
Furthermore, KAIST student examination committee will officially punish those who deny their cheating activities (i.e., suspension, expulsion, 정학, 퇴학)

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

<http://cs101.kaist.ac.kr>

- Lecture slides and sample codes
- Lecture notes on robot programming and photo processing
- Lab materials

Main reference:



Practical Programming: An Introduction to Computer Science Using Python 3.6 (3rd Edition) by Paul Gries, Jennifer Campbell, and Jason Montojo, Pragmatic Bookshelf, 2017, ISBN 978-1680502688

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

- Used in many universities for introductory courses
- Main language used for artificial intelligence (AI) at Google, Meta, OpenAI
- e.g., Kaggle competitions

There's no future for traders who don't know Python

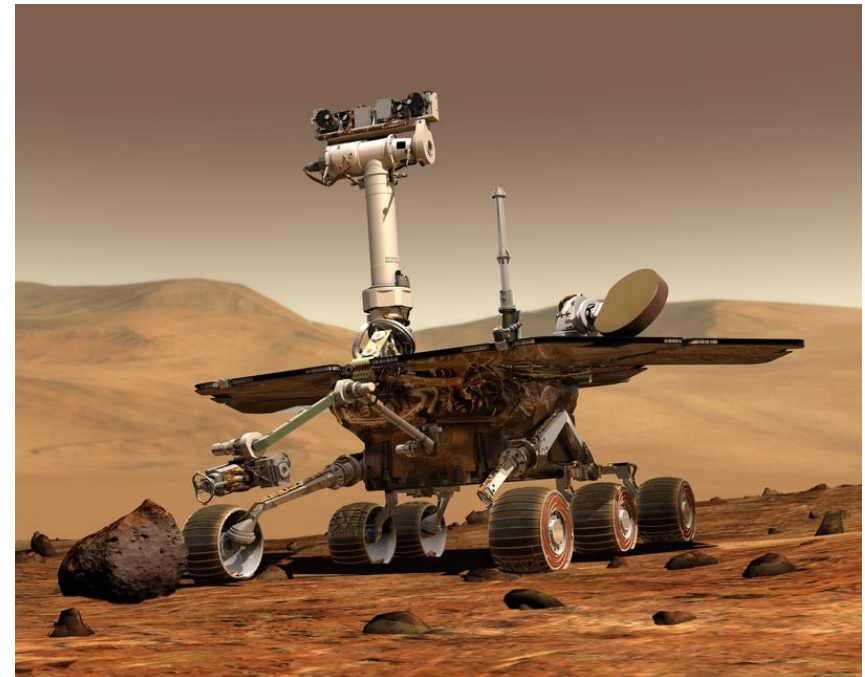
by Sarah Butcher 20 August 2019



<https://news.efinancialcareers.com/no-en/3001881/learn-python-traders-in-banks>

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

- Widely used in scientific computation, for instance at NASA, by mathematicians and physicists
 - Many NASA Open Source Software Projects are done with Python (<https://code.nasa.gov/?q=python>)
 - Texting robots on Mars using Python, Flask, NASA APIs and Twilio MMS (<https://www.twilio.com/blog/2017/04/texting-robots-on-mars-using-python-flask-nasa-apis-and-twilio-mms.html>)



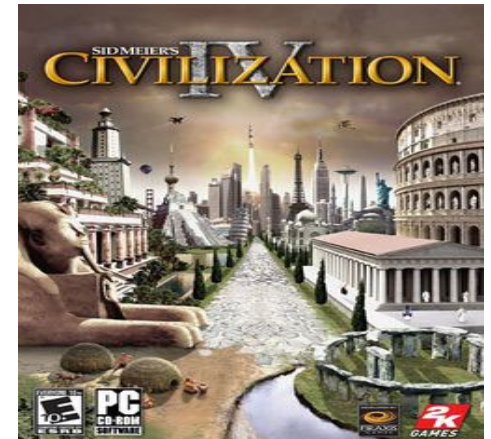
Python is a programming language that is **easy to learn** and very powerful.

- Available on embedded platforms, for instance Nokia mobile phones.
 - e.g., Library for the Nokia Health API (<https://github.com/orcasgit/python-nokia>)



- Large portions of games (such as Civilization IV) are written in Python.
 - “Python can be used to modify the behavior of Civilization IV, and is essential if you want to create a new game. This tutorial for python originated from the Civilization Fanatics Center forum, and will provide basic information on how to modify the game.”
(https://strategywiki.org/wiki/Civilization_IV/Python_Tutorial)

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

https://en.wikipedia.org/wiki/Alan_Perlis



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?

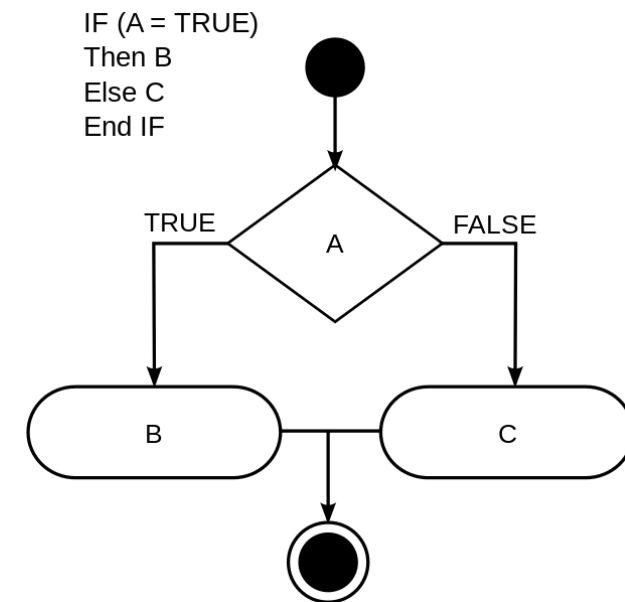
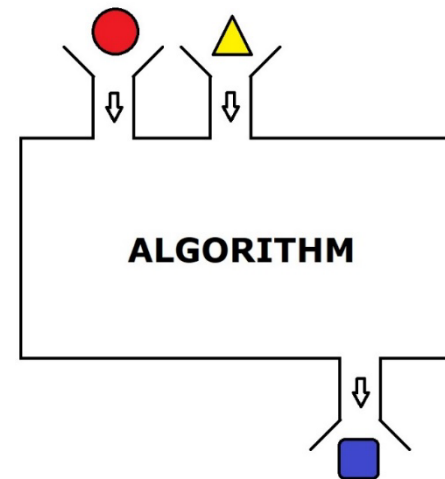
A program is a **sequence of instructions** that solves some problem (or achieves some effect).

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).

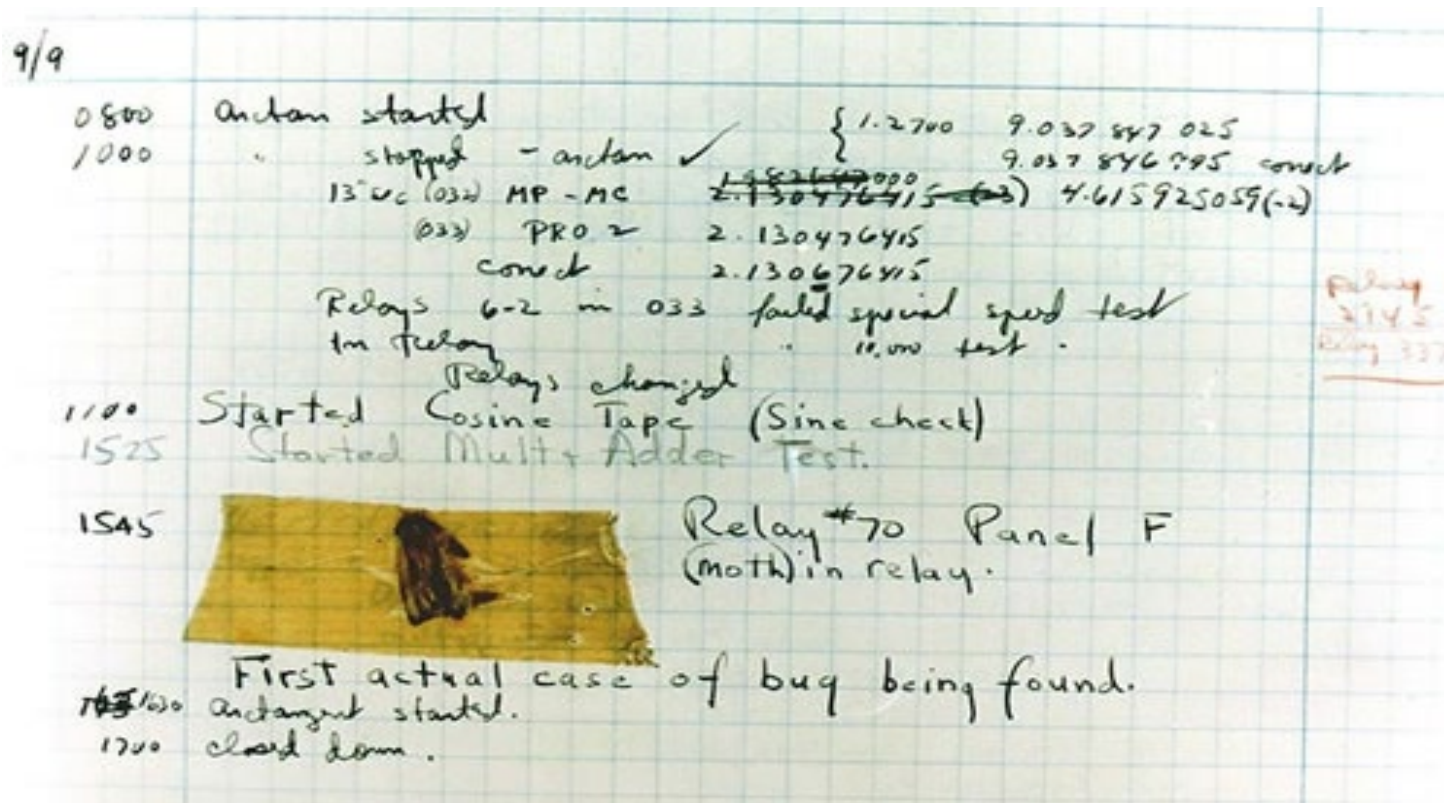


What is debugging?

A **bug** is a mistake in a program.

- On September 9, 1945, U.S. Navy officer Grace Hopper found a moth between the relays on the Harvard Mark II computer she was working on

Debugging means to find the mistake and to fix it.



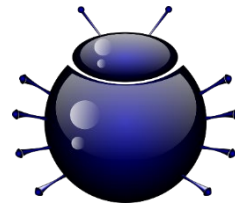
The very first recorded computer bug

<https://thenextweb.com/shareables/2013/09/18/the-very-first-computer-bug/>

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.

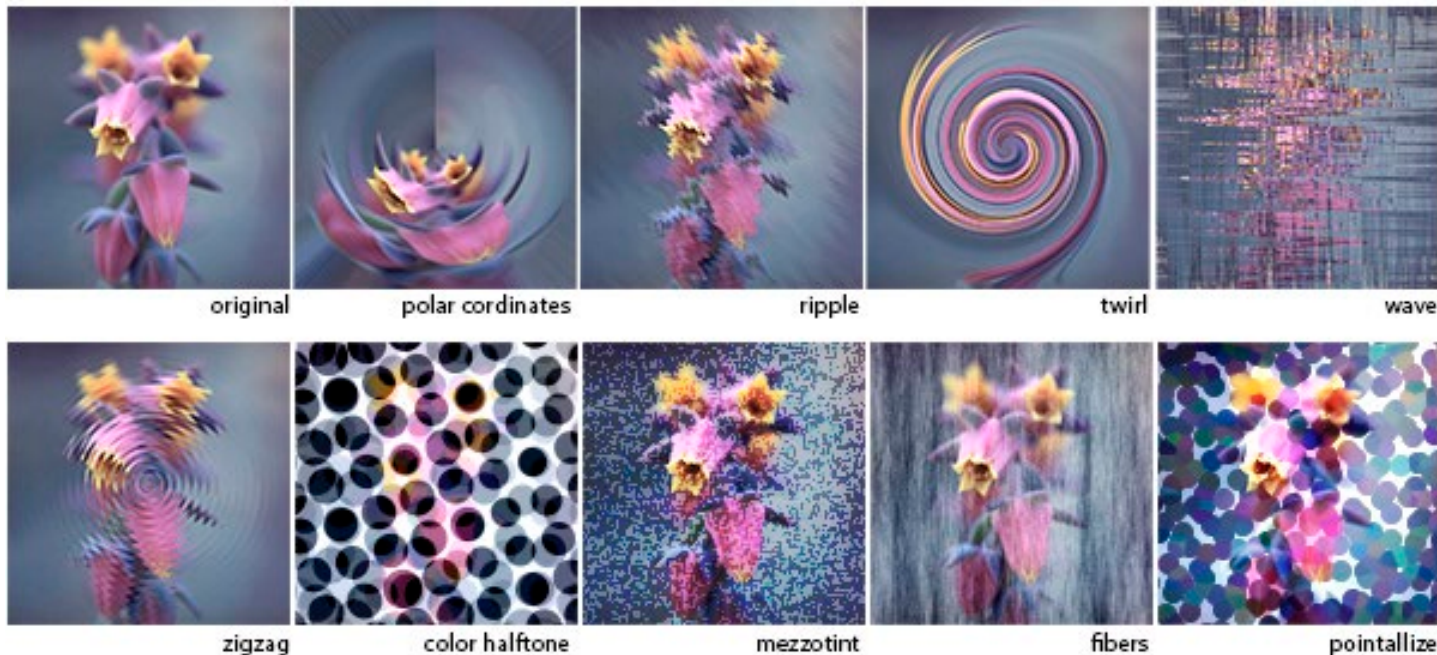
Why is programming useful?

- 20 years ago, **electrical engineering students** learned about circuits. Today they learn about embedded systems.
- **Industrial engineers** program industrial robots. Moreover, today's industrial engineers work on logistics/problems that can only be solved by computer.
- **Modern automobiles** contain millions 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.

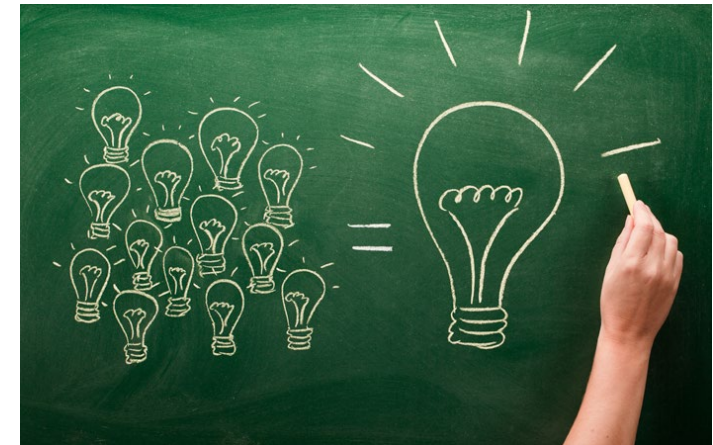
A single person can actually build a software system of the complexity of the space shuttle hardware. Nothing similar is true in any other discipline.

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.



GitHub

<https://github.com/>



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

What is Elice?

Online programming education platform

In CS101, we will use them for

- Programming tasks in lab
- Homework assignments



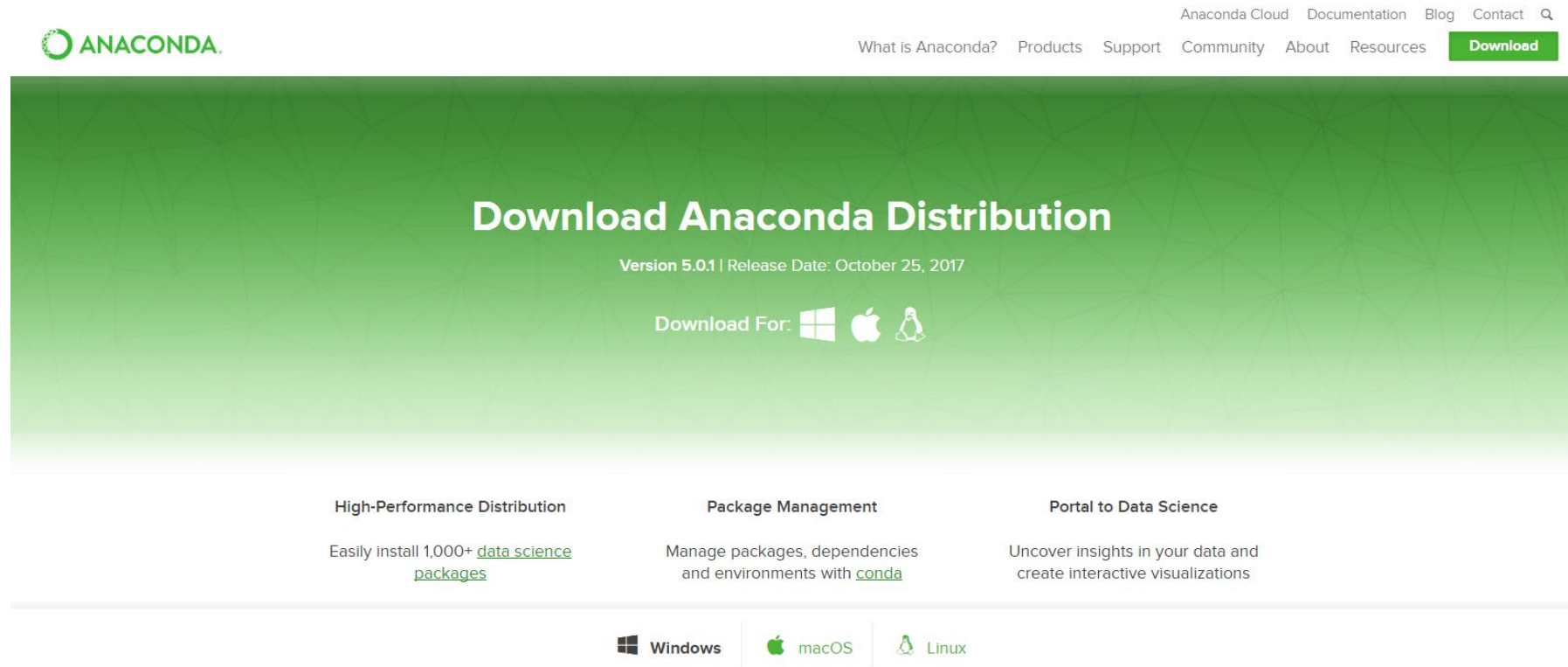
No need to spend a lot of time setting up your development environment for CS101!

Focus fully on your ideas & writing your code.

Students should not use Elice's AI Helper capability.

Anaconda (Optional)

- Professor's choice: Anaconda
 - Download and install
<https://docs.anaconda.com/anaconda/install/>
 - It's free!



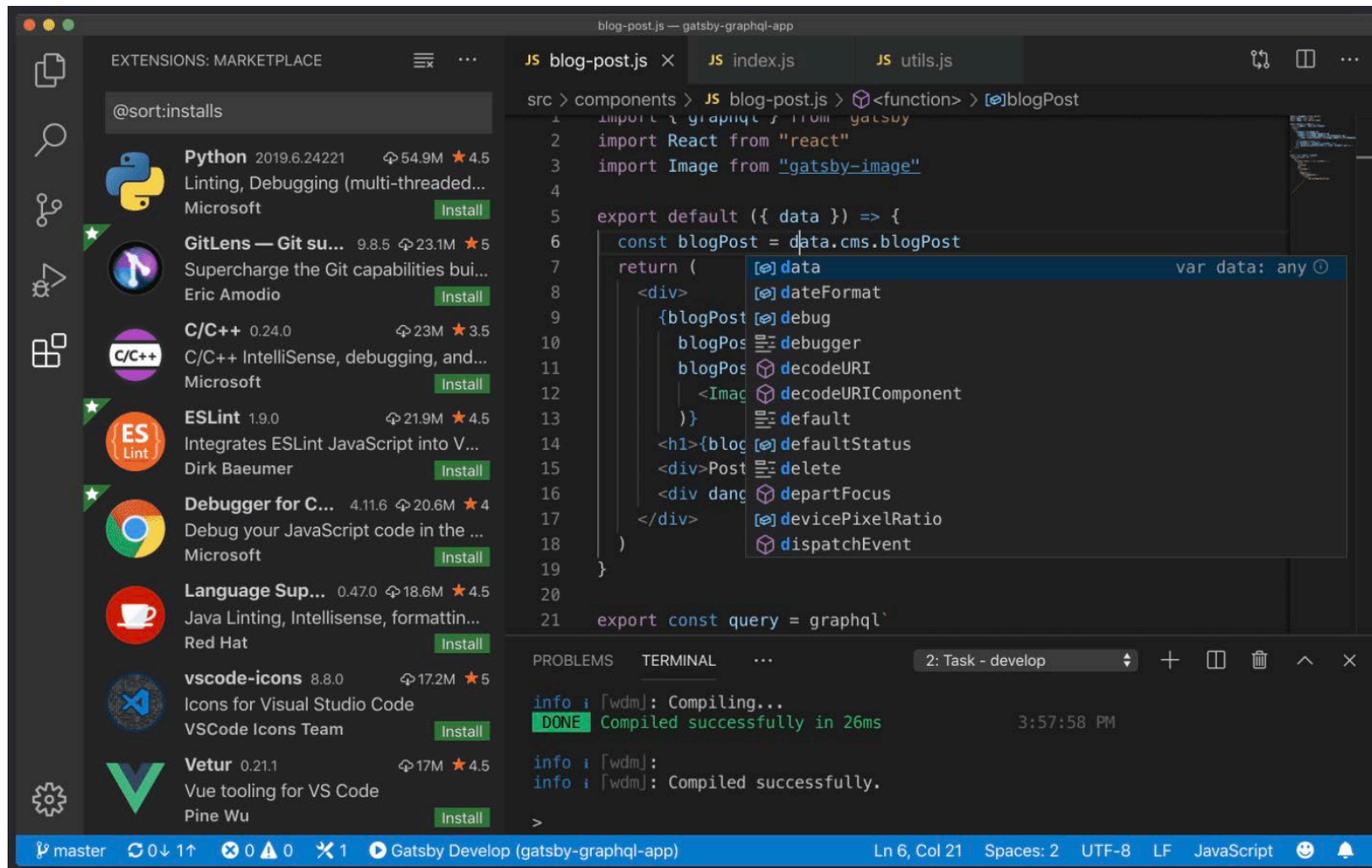
Visual Studio Code IDE Edu (Optional)

- Professor's choice: Visual Studio Code

- Download and install

<https://code.visualstudio.com/download>

- It's free!

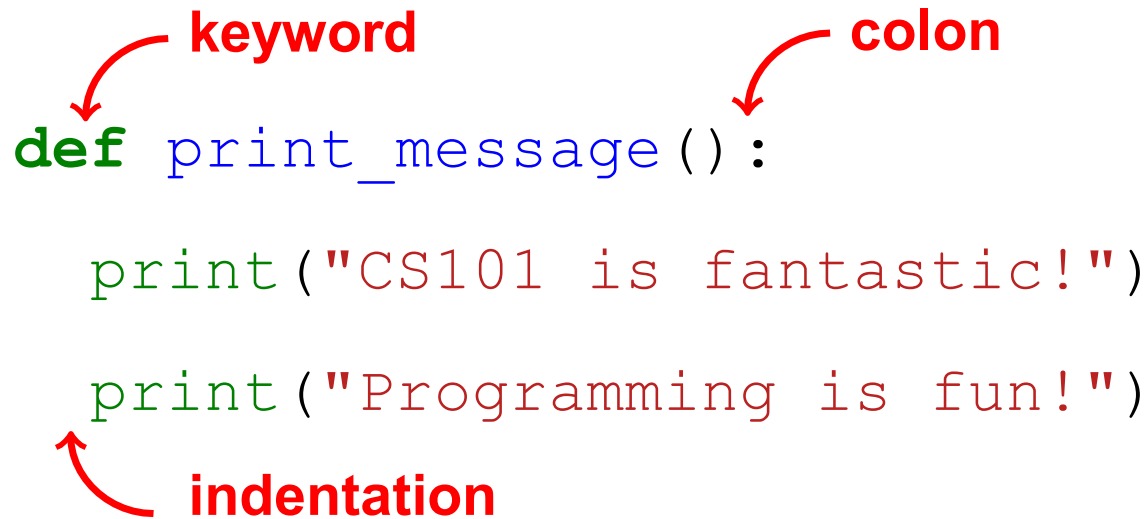


Install CS101 Libraries (Optional)

- Download CS101 Libraries from Elice
- Find out the site-package path in your Python
 - `python -m site`
 - `/Users/YOURID/opt/anaconda3/lib/python3.8/site-packages`
- Copy all the files to the folder
 - `cs1elice.py`, `cs1media.py`, `cs1robots_images.py`,
`cs1graphics.py`, `cs1robots.py`, `easygui.py`

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**.



The diagram illustrates the syntax of a function definition in Python. It shows the code: `def print_message():` followed by two indented lines: `print("CS101 is fantastic!")` and `print("Programming is fun!")`. Red arrows point to specific parts of the code with labels: an arrow points to `def` with the label **keyword**, another arrow points to the colon `:` with the label **colon**, and a third arrow points to the indentation of the first line with the label **indentation**.

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

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()
```

function definitions

function calls

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.

```
# create a robot with one beeper
```

```
hubo = Robot(bepers = 1)
```

object with default parameters

```
# move one step forward
```

```
hubo.move()
```

method: dot notation

```
# turn left 90 degrees
```

```
hubo.turn_left()
```

How can hubo turn right?

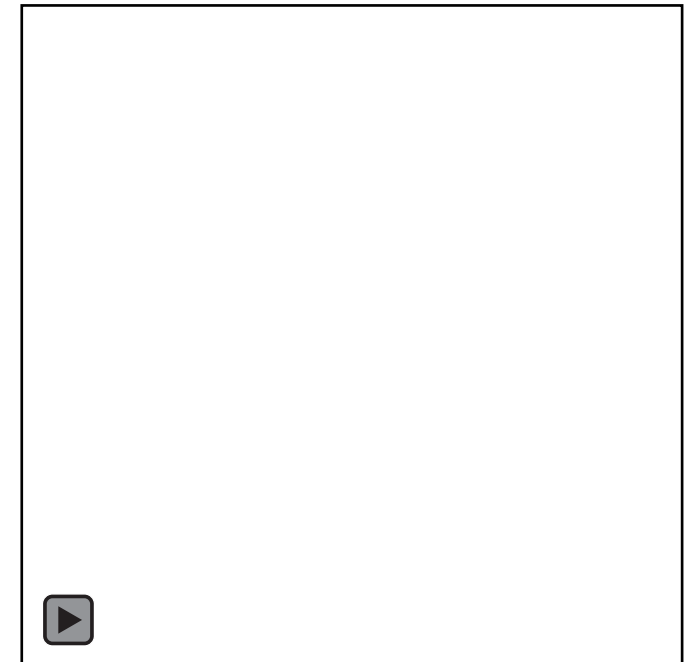
```
def turn_right():
```

```
    hubo.turn_left()
```

```
    hubo.turn_left()
```

```
    hubo.turn_left()
```

Define a function!



Top-down design

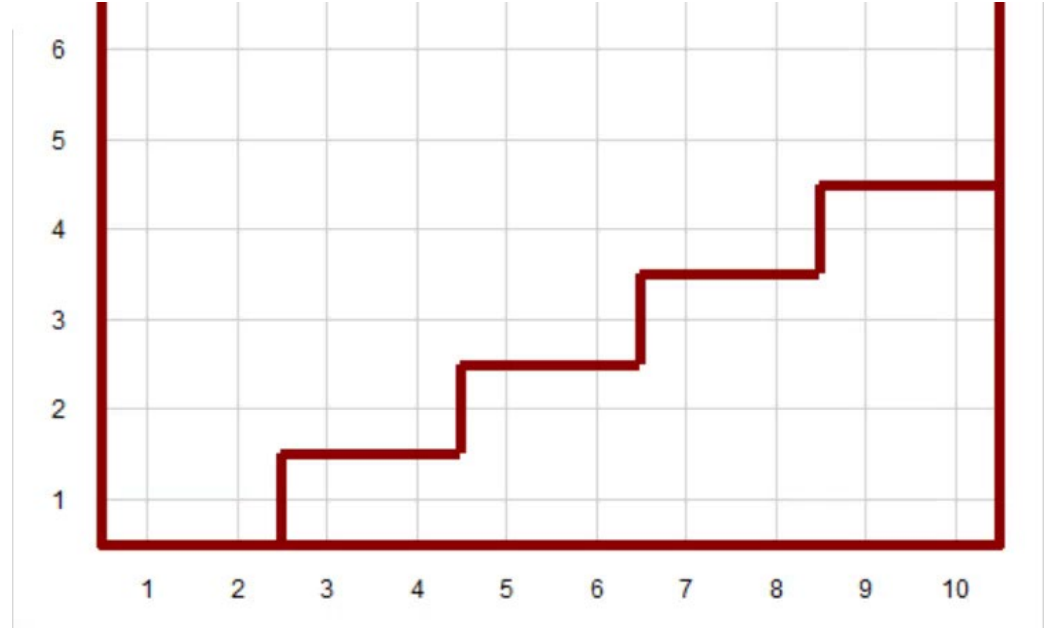
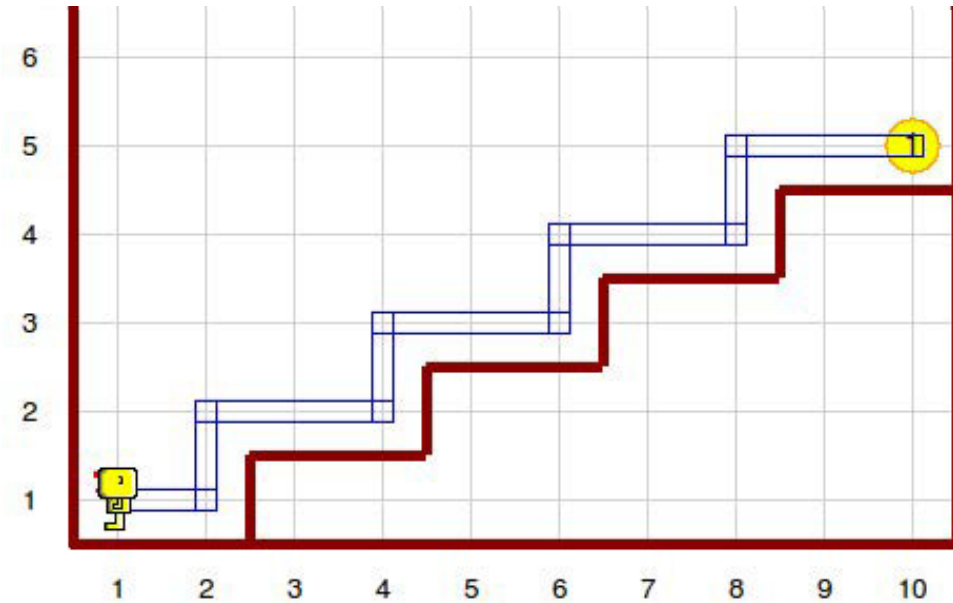
Start at the top of the problem, and make an outline of a solution.

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.

Example: 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()
```

Implementing the steps

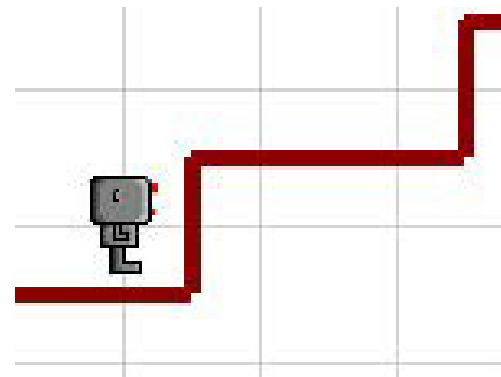
```
def turn_around():  
    hubo.turn_left()  
    hubo.turn_left()  
  
def climb_up_four_stairs():  
    # how?
```

Implementing the steps

```
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()
```



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()
```

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()
```


Questions?