Review

- Data Science
  - Drawing useful **conclusions** from data using **computation**

- Computer Science
  - Making computers do what you want to do **efficiently** (for your **productivity**)

- Algorithms and Programs
  - **Logical** steps (recipe)
  - Written in a **computer language** (instruction set)

Abstraction

Lecture 1-1

Hyung-Sin Kim

SNU Graduate School of Data Science

How does a computer run a Python program?

Abstraction – Car

Interface

Don’t worry about it ☺

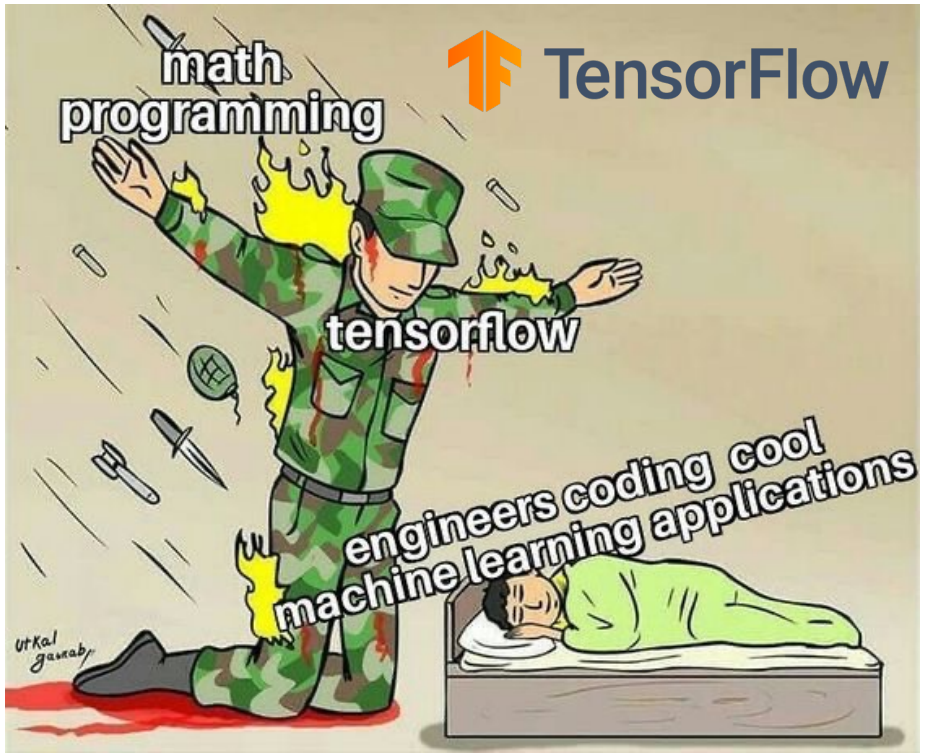
Abstraction

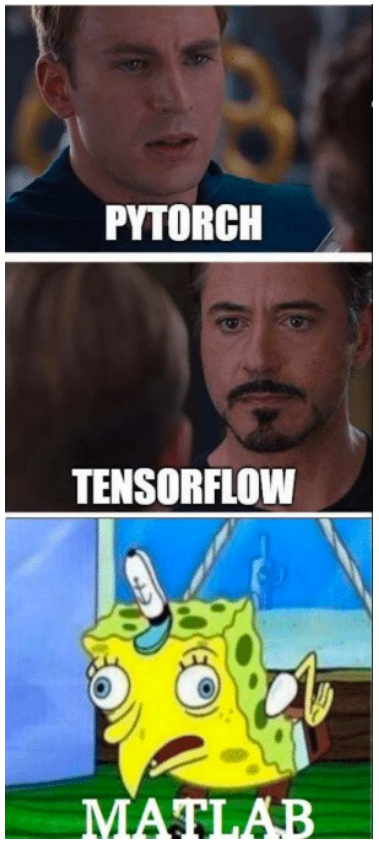
- Drivers do not need to know **implementation** details about a car to drive it
  - Tire, wheel, engine, size, weight

- They just need to deal with user **interfaces**, such as handle and pedals
  - Once they know how to handle interfaces, they can drive all of various cars in the world

- **Abstraction**: The process of preserving information that is relevant in a given context, and forgetting information that is irrelevant in that context
  - We cannot remember and focus on many things at a time …
  - People with different expertise/interest can focus on their jobs without worrying about other part

Abstraction – Machine Learning





https://www.youtube.com/watch?v=CS4cs9xVecg&list=PLkDaE6sCZn6Ec-XTbcX1uRg2_u4xOEky0

Abstraction – Mobile Phone

Application programmers do not need to know various hardware details thanks to…

Abstraction – Mobile Phone

Software

Application





Operating System

Interface

android

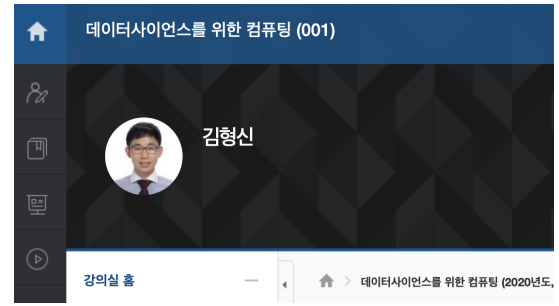
Implementation

Hardware

Abstraction – More General Computer

Software

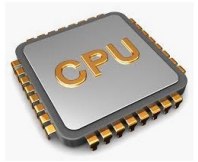
Application

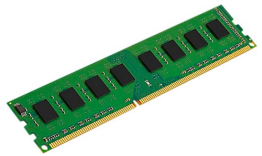


Operating System

Application programmers do not need to know various hardware details thanks to…

Hardware

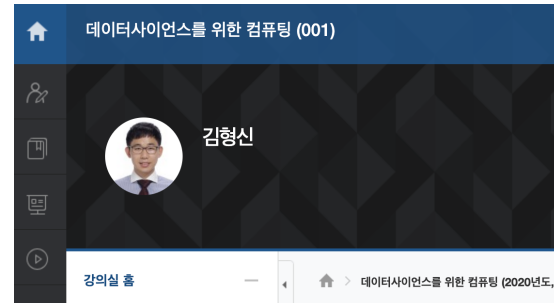




Abstraction – More General Computer

Software

Application



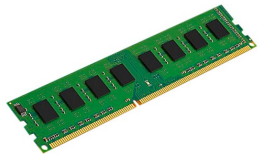
Interface

Operating System

Windows 10

Implementation

Hardware



Abstraction – Thank You Operating System!

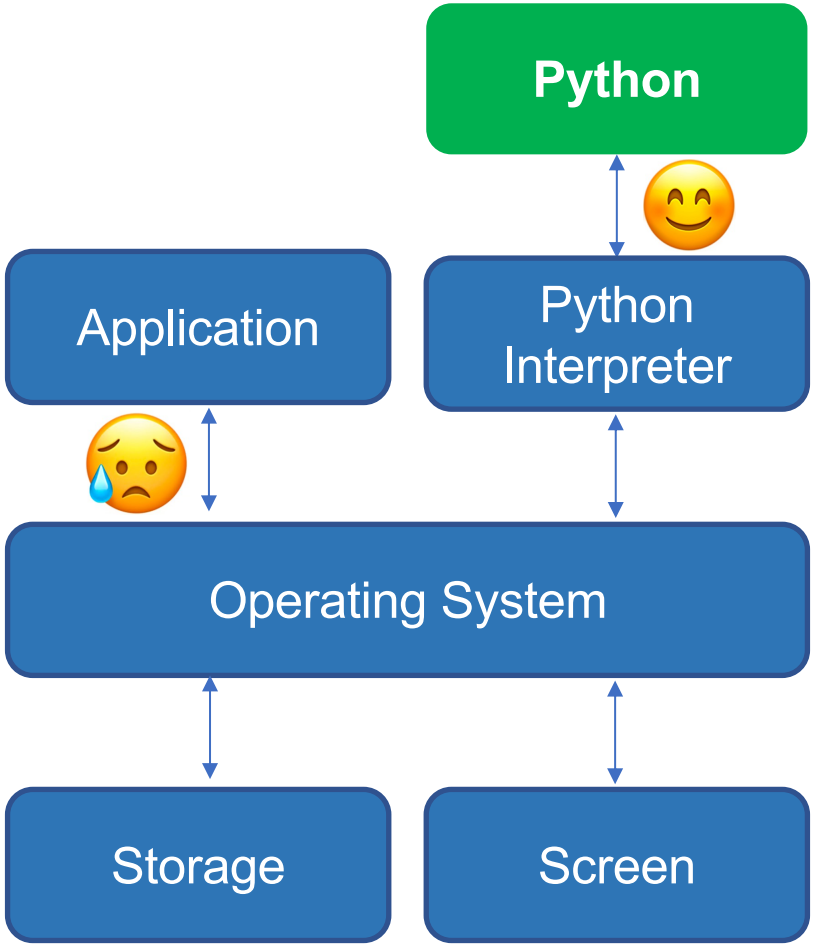
- Operating system (e.g., MS Windows, Linux, macOS) is the only program on the computer that's allowed **direct access** to the hardware

- If any other application program wants to interact with hardware (fetch data from storage, draw on the screen, find out what key was just pressed on the keyboard), it sends a **request** to the OS (**indirect access**)

- If you are not writing an OS, you don't need to care about hardware but OS
  - Some smartphone Apps support either Android or iOS, not both of them
  - Once an app is ported on Android, it works for all different kinds of Android phones

Abstraction – Thank You Interpreter!

- OS is still too low and complex to directly program an application on…
  - If you are not a hardcore programmer
- Another layer for you, called **Interpreter**
  - If you write a program using Python, the interpreter takes your program, translate it into a language that OS understands
- Now you don't have to worry about OS too! Just write a program and it will run on various OSes!

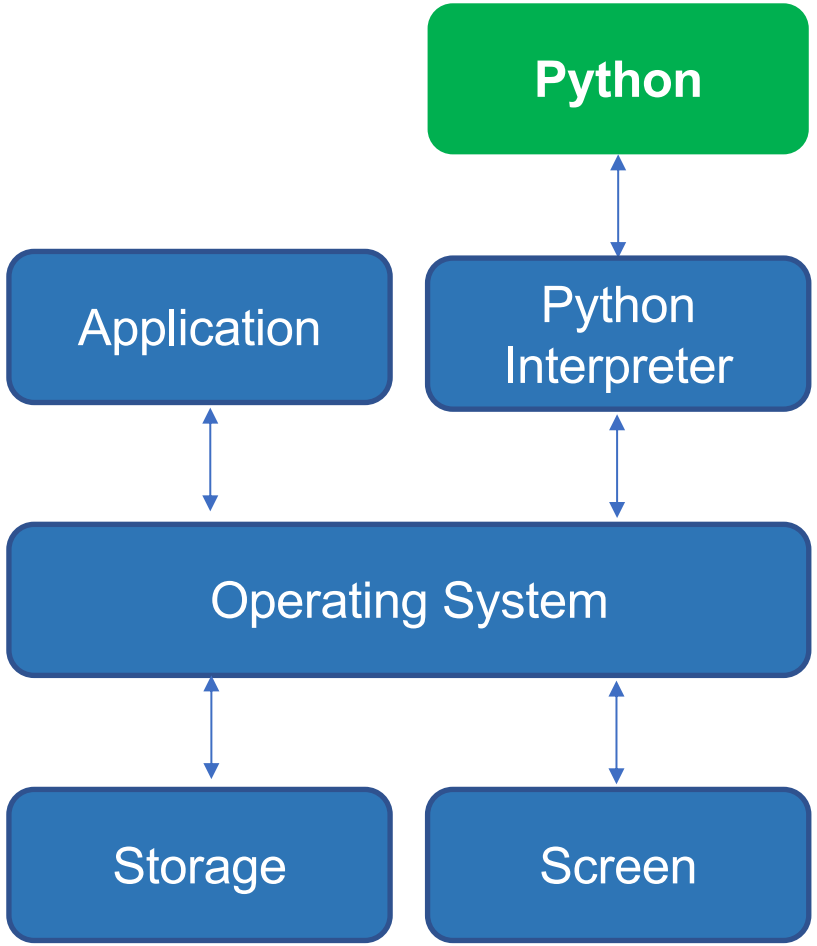


Python Interpreter

- Multiple ways of playing with Python
  - Execute a Python program that is saved in a file with a **.py** extension
    - The interpreter will execute **the whole program** in the file
  - Execute a program called a **shell**, and type Python statements one at a time
    - The interpreter will execute **each statement** when you type it
  - Execute **jupyter notebook**, type and execute a group of Python statements
    - The interpreter will execute **the group of statements**
- With the Python interpreter, you only care about Python, neither OS nor hardware

Summary

- Abstraction – Interface and implementation
  - For now, it is OK to know only interfaces
  - To become a power user, you need to learn **low-level system programming**!
- Operating system and application
- Python Interpreter
- This figure



Hello, Python!

Lecture 1-2

Hyung-Sin Kim

SNU Graduate School of Data Science

Let's start programming!

Your First Program

- Open a Jupyter cell

- Let's type a simple mathematical expression and execute the code (press Ctrl+Enter or click Run)
  - >>> 3 + 4
  - 7

- Congratulation! You wrote a code and it works!

Primitive Expressions

- What you just typed is an **expression** including
  - **Operators**: +, -, *, /, %, //, **
  - **Operands**: Values that an operator takes

- An expression does not have to have an operator, a single value is also an expression

- **Evaluation**: When you type an expression and run it, Python evaluates the expression, produces a value, and shows it

Primitive Expressions

- Operators precedence
  - **
  - - (negation)
  - *, /, //, %
  - +, -
- Example
  - -2 ** 4 ⟹ -16
  - -(2 ** 4) ⟹ -16
  - (-2) ** 4 ⟹ 16

- Type whatever mathematical expressions for fun and see how Python evaluates them!

Types

- Every value in Python has a particular **type**
  - *int* (integer): 1, 4, 8, 10, 100 …
  - *float* (floating point): 2.5, 19.2, 7.1 …

- An expression having two *floats* produces a float
  - 100.0 – 25.0 ⟹ 75.0

- An expression having an *int* and a *float* makes Python convert the *int* to a *float*
  - 100 - 25.0 ⟹ 75.0

Types

- A type in Python consists of two things
  - A set of values
  - A set of operations that can be applied to those values

- **int**: (1) integer numbers, (2) arithmetic operators can be applied
- **float**: (1) a subset of the real numbers, (2) arithmetic operators can be applied

- Finite precision: Computer does not represent all the real numbers due to its limited memory, representing the closest value it can produce
  - 2 / 3 ⟹ 0.666666666666
  - This is memory efficient and allows fast calculation

Call Expressions

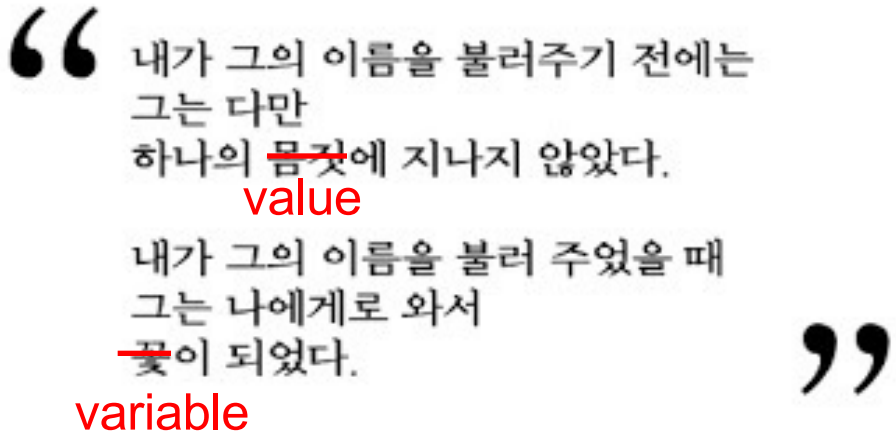
- Operator can be a function (there are many functions that Python provides)
- Call expression: An expression that calls a function
- Example
  - max(2,3)
  - **Operators**: max – a function name
  - **Operands**: 2 and 3 – again, values that an operator takes
- An operand can also be a call expression
  - max(min(2, 3) , min(9, 10))

Variables and Names

- **Values** do not have any meaning, so now we want to **name** them: **Variable**

- Name
  - Letters, digits, and the underscore symbol
  - Cannot start with a digit: ~~7ab~~
  - Case sensitive: GSDS vs. gsds
  - No empty space
  - Naming properly is very important!

- Reserved word (or keyword)
  - A name that Python already uses: True, False, if, for …
  - You cannot use these words as your own names



Assignment

- You can create a new variable by **naming** it and **assigning** it a value
  - temp_celsius = 31.0 (**assignment statement**)

  - "temp_celsius is assigned the value 31.0."
  - "=" is not equality!

- You can do assignment for multiple variables on a single line
  - x, y = 1, 2
  - y, x = x, y

Assignment

- When Python sees a variable in an expression, it uses its assigned value
  - temp_celsius + 5/10 ⟹ 31.5

- We can **reassign** another value to an existing variable (yes, it is "**variable**"!)
  - temp_celsius = -15.2

Summary

- Expression and evaluation
- Operator and operand
- Value and type
- Name and variable

- Assignment

Memory Model and Reassignment

Lecture 1-3

Hyung-Sin Kim

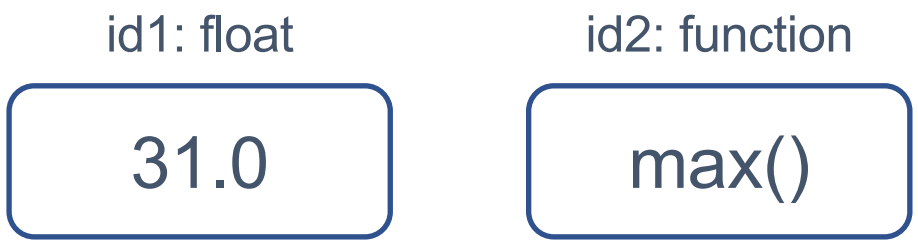
SNU Graduate School of Data Science

What happens in the computer
when you execute an assignment statement?

Memory Model

- A memory object
  - Address
  - Value

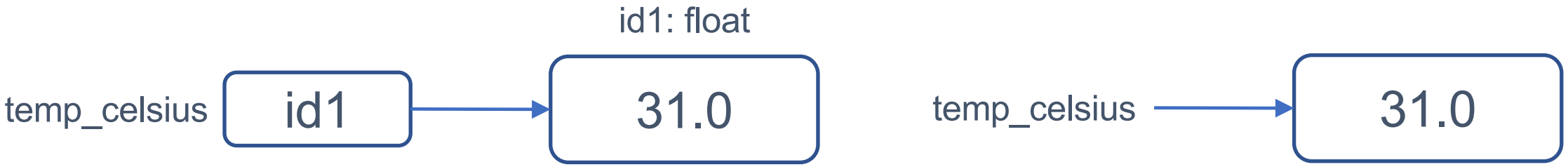
- Example
  - The object at the memory address id1 has type float and the value 31.0
  - The object at the memory address id2 has type function and max



Memory Model – Assignment

- <<variable>> = <<expression>>
  - Step 1: Evaluate the expression on the right side to produce a value. This value is stored in a memory object
  - Step 2: Store the **address** of the memory object (containing the value above) in the variable on the left side
    - If the variable already exists, replace the memory address that it contains
  - Result: the **variable** points the memory where the **value** is stored

- Example: temp_celsius = 31.0

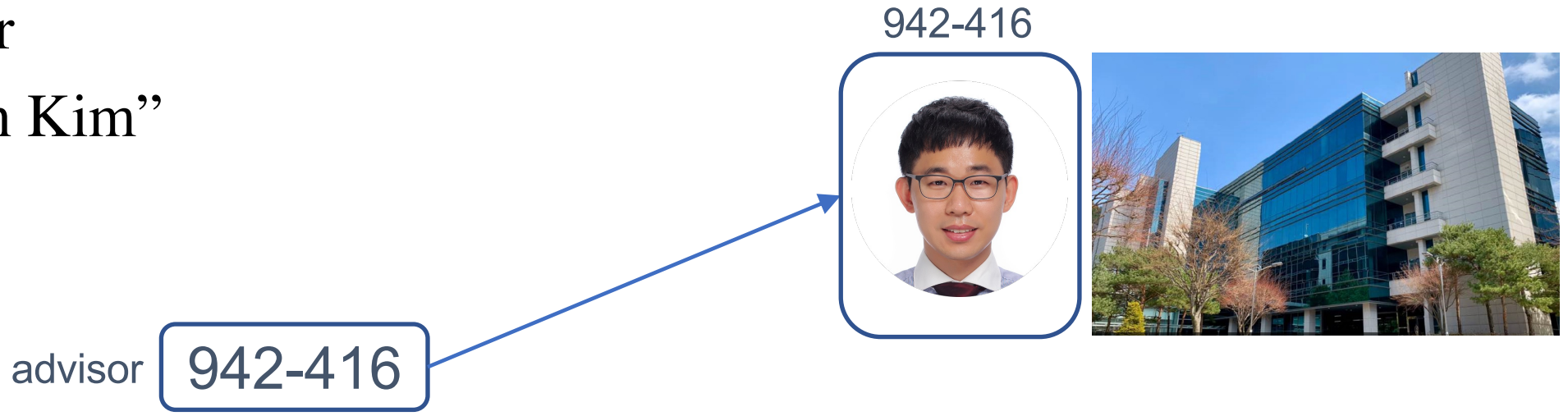


Memory Model – Assignment

- <<var1>>, <<var2>>, …, <<varN>> = <<exp1>>, <<exp2>>, …, <<expN>>

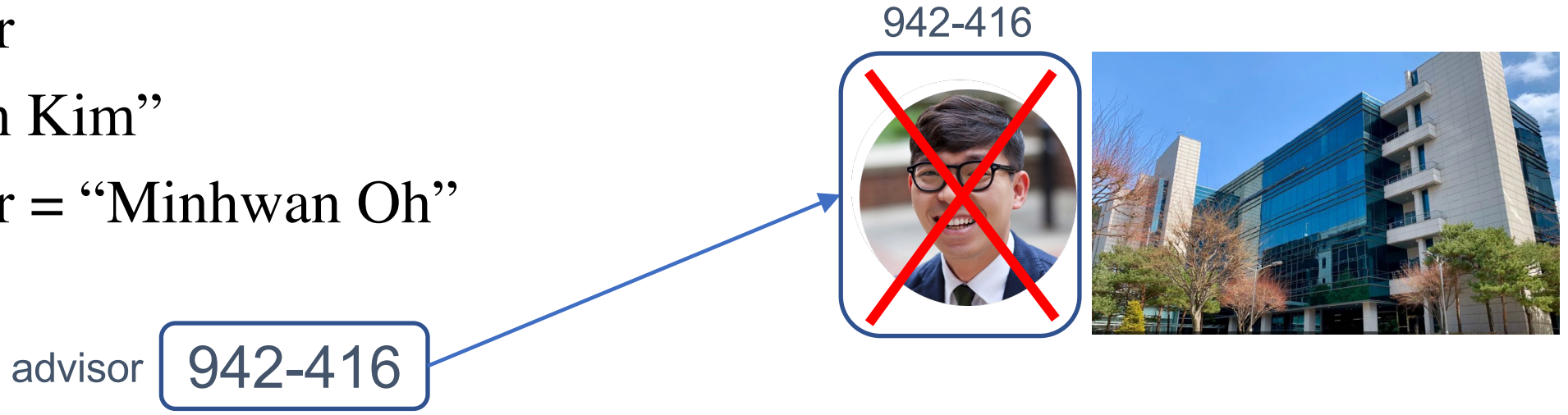
Memory Model – Reassignment (1)

- >>> advisor = “Hyung-Sin Kim”
- >>> advisor
- “Hyung-Sin Kim”



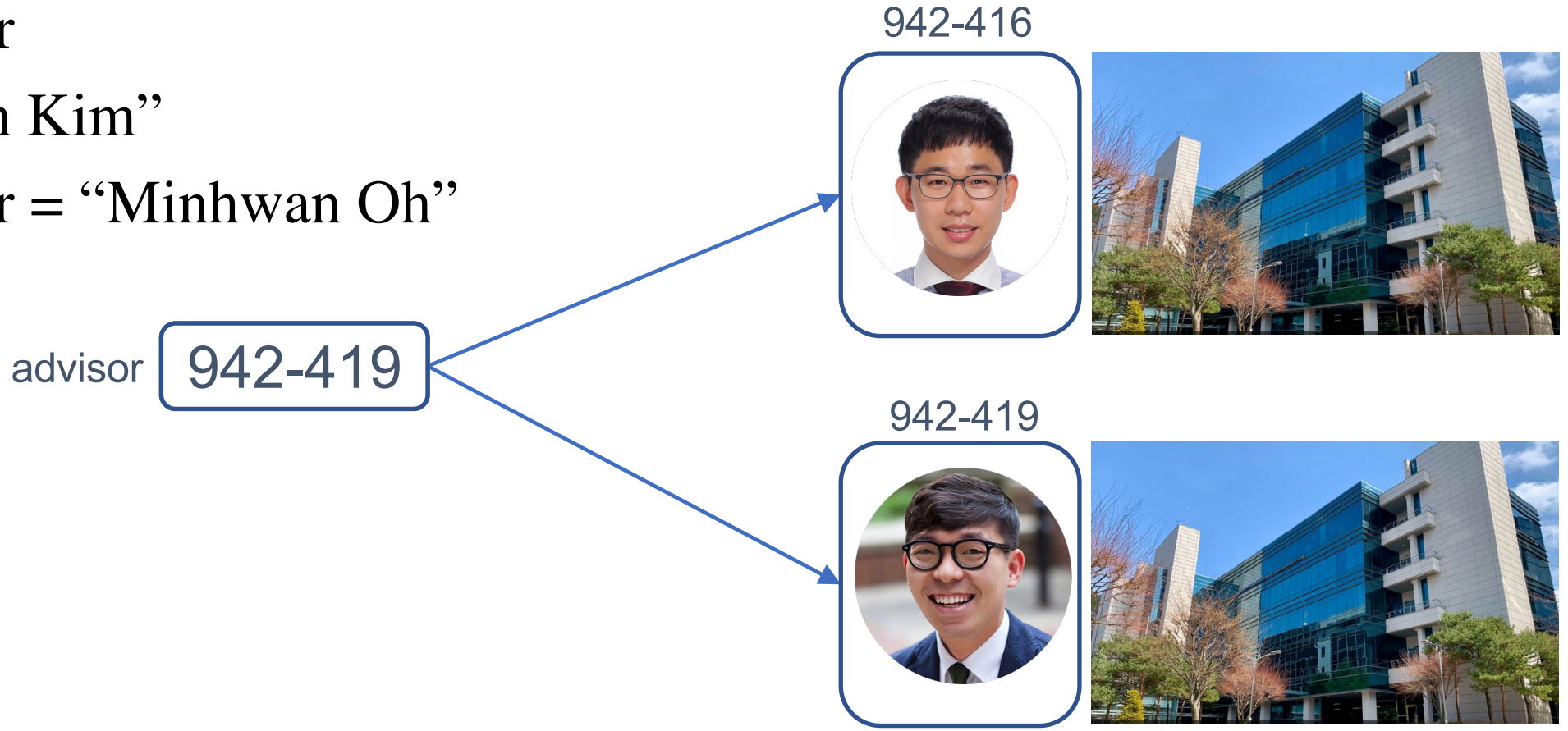
Memory Model – Reassignment (1)

- >>> advisor = “Hyung-Sin Kim”
- >>> advisor
- “Hyung-Sin Kim”
- >>> advisor = “Minhwan Oh”



Memory Model – Reassignment (1)

- >>> advisor = “Hyung-Sin Kim”
- >>> advisor
- “Hyung-Sin Kim”
- >>> advisor = “Minhwan Oh”



Memory Model – Reassignment (2)

- >>> temp_celsius = 31.0
- >>> difference = 10.0
- >>> temp_celsius = temp_celsius – 2*difference
- >>> difference = 5.0
- >>> temp_celsius
- ?

Memory Model – Reassignment (2)

- >>> temp_celsius = **31.0**
- >>> difference = 10.0
- >>> temp_celsius = temp_celsius – 2*difference
- >>> difference = 5.0
- >>> temp_celsius
- ?

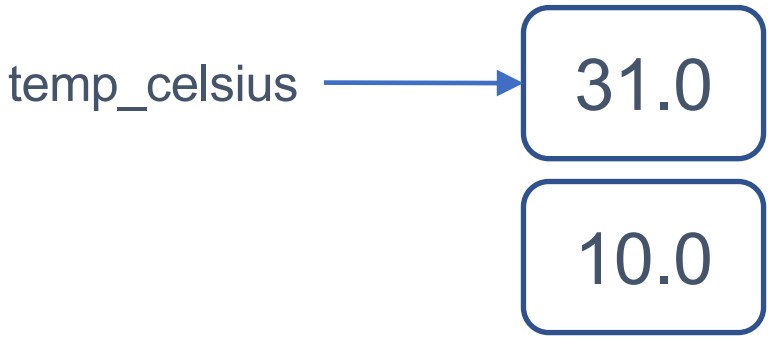
31.0

Memory Model – Reassignment (2)

- >>> **temp_celsius =** 31.0
- >>> difference = 10.0
- >>> temp_celsius = temp_celsius – 2*difference
- >>> difference = 5.0
- >>> temp_celsius
- ?

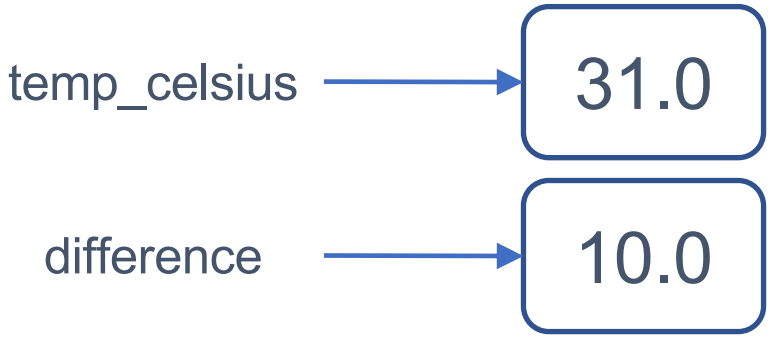
Memory Model – Reassignment (2)

- >>> temp_celsius = 31.0
- >>> difference = **10.0**
- >>> temp_celsius = temp_celsius – 2*difference
- >>> difference = 5.0
- >>> temp_celsius
- ?



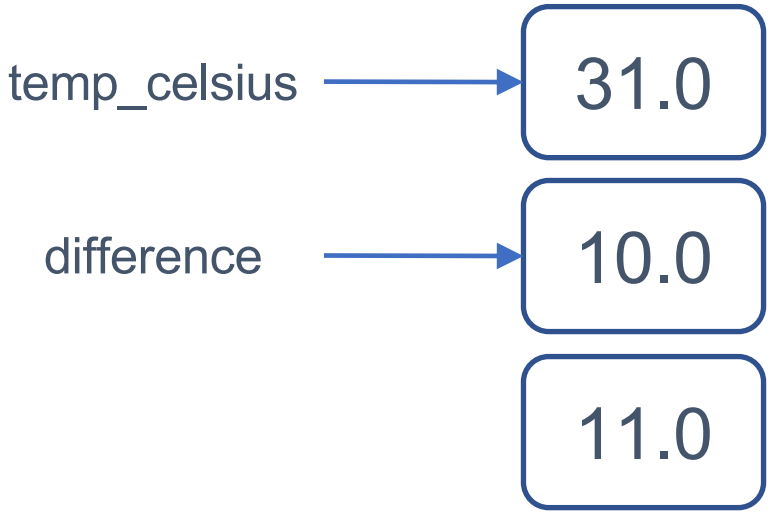
Memory Model – Reassignment (2)

- >>> temp_celsius = 31.0
- >>> **difference =** 10.0
- >>> temp_celsius = temp_celsius – 2*difference
- >>> difference = 5.0
- >>> temp_celsius
- ?



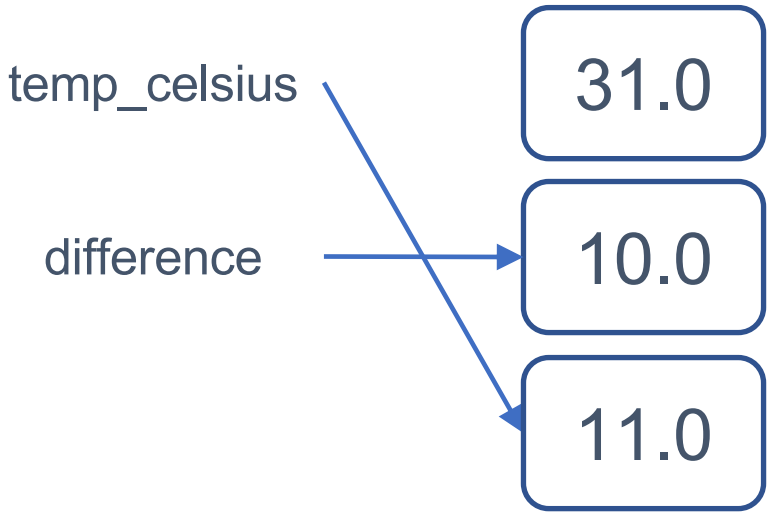
Memory Model – Reassignment (2)

- >>> temp_celsius = 31.0
- >>> difference = 10.0
- >>> temp_celsius = **temp_celsius – 2*difference**
- >>> difference = 5.0
- >>> temp_celsius
- ?



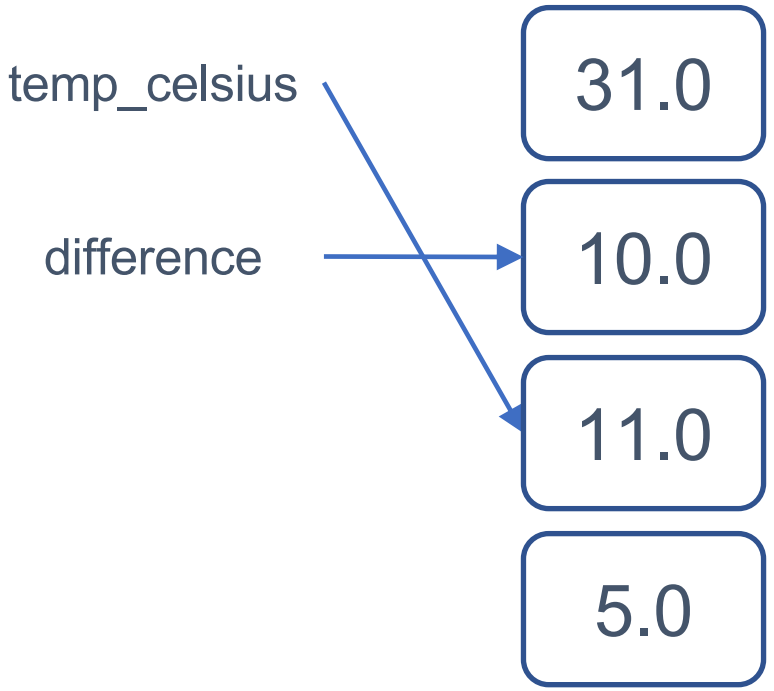
Memory Model – Reassignment (2)

- >>> temp_celsius = 31.0
- >>> difference = 10.0
- >>> **temp_celsius =** temp_celsius – 2*difference
- >>> difference = 5.0
- >>> temp_celsius
- ?



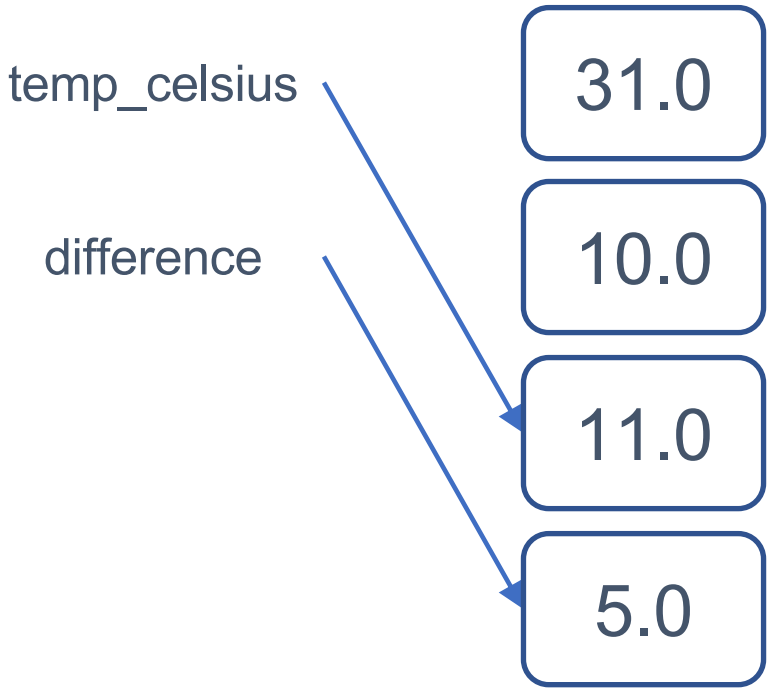
Memory Model – Reassignment (2)

- >>> temp_celsius = 31.0
- >>> difference = 10.0
- >>> temp_celsius = temp_celsius – 2*difference
- >>> difference = **5.0**
- >>> temp_celsius
- ?



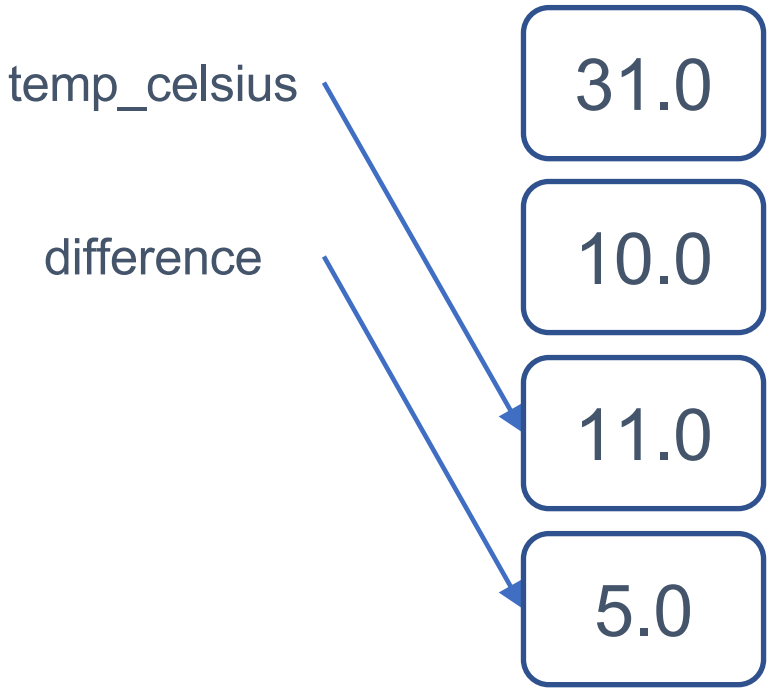
Memory Model – Reassignment (2)

- >>> temp_celsius = 31.0
- >>> difference = 10.0
- >>> temp_celsius = temp_celsius – 2*difference
- >>> **difference =** 5.0
- >>> temp_celsius
- ?



Memory Model – Reassignment (2)

- >>> temp_celsius = 31.0
- >>> difference = 10.0
- >>> temp_celsius = temp_celsius – 2*difference
- >>> difference = 5.0
- >>> temp_celsius
- ?
- **11.0**



Summary

- Memory model
- Each value dwells in a memory box
- A variable points at the memory box where its assigned values lives
- Reassignment is pointing at a different memory box

Thanks!