CE 311K: Introduction to Computer Methods

Krishna Kumar

University of Texas at Austin

krishnak@utexas.edu

August 11, 2019

Overview

1. Simulations
2. Aspects of languages
3. Python
4. Numerical solution
   - Numerical solution of a sliding block

On computable numbers

Alan Turing

3-Rotor Enigma

The Bombe

Enigma has 150,738,274,937,250 possible states. (credit: Rutherford journal)

CE 311K: Intro to Computer Methods

On computable numbers

Alan Turing 3-Rotor Enigma The Bombe

Enigma has 150,738,274,937,250 possible states. (credit: Rutherford journal)

Turing began to consider whether a method or process could be devised that could decide whether a given mathematical assertion was provable. Turing analyzed the methodical process, focusing on logical instructions, the action of the mind, and a machine that could be embodied as a physical form. Turing developed the proof that automatic computation cannot solve all mathematical problems. This concept became known as the Turing machine, which has become the foundation of the modern theory of computation and computability.

The Prophet and the Pioneer Computer Scientists

Ada Lovelace, circa 1838 (credit: Science Museum, California)

Grace Hopper (credit: unknown)

CE 311K: Intro to Computer Methods

The Prophet and the Pioneer Computer Scientists

Ada Lovelace, circa 1838 (credit: Science Museum, California)

Grace Hopper (credit: unknown)

Lovelace speculated that the Engine 'might act upon other things besides number... the Engine might compose elaborate and scientific pieces of music of any degree of complexity or extent'. The idea of a machine that could manipulate symbols in accordance with rules and that number could represent entities other than quantity mark the fundamental transition from calculation to computation.

Hopper is the pioneer computer scientist, known for creating COBOL – the first computing language.

To infinity and beyond...

Margaret Hamilton next to a stack of the Apollo Guidance Computer source code (1969, credit: MIT Museum) and Katie Bouman who developed the algorithm for creating the first-ever image of black hole (2019, credits: PBS).

? Could you guess the storage size requirements?

CE 311K: Intro to Computer Methods

To infinity and beyond...

To infinity and beyond...

Margaret Hamilton next to a stack of the Apollo Guidance Computer source code (1969, credit: MIT Museum) and Katie Bouman who developed the algorithm for creating the first-ever image of black hole (2019, credits: PBS).

Could you guess the storage size requirements?

Margaret Hamilton next to a stack of the Apollo Guidance Computer source code (1969, with 11,000 lines of code running on **72 kilobytes** of computer memory, credits: MIT Museum) and Katie Bouman who developed the algorithm for creating the first-ever image of black hole (2019, credits: PBS) and the HDD of **5 petabytes** of data.

If we assume a typical MP3 song as 4MB, we can fit 56 Apollo guidance code in a song, while the 5 PetaBytes of data can hold 250 million songs (there are roughly 97 million songs in the world.)

To infinity and beyond...

CE 311K: Intro to Computer Methods

To infinity and beyond...

Scientists have obtained the first image of a black hole, using Event Horizon Telescope observations of the center of the galaxy M87. The image shows a bright ring formed as light bends in the intense gravity around a black hole that is 6.5 billion times more massive than the Sun. The task of observing this super massive black hole is extremely hard, the size ratio is same as observing an orange on the surface of moon from the Earth.

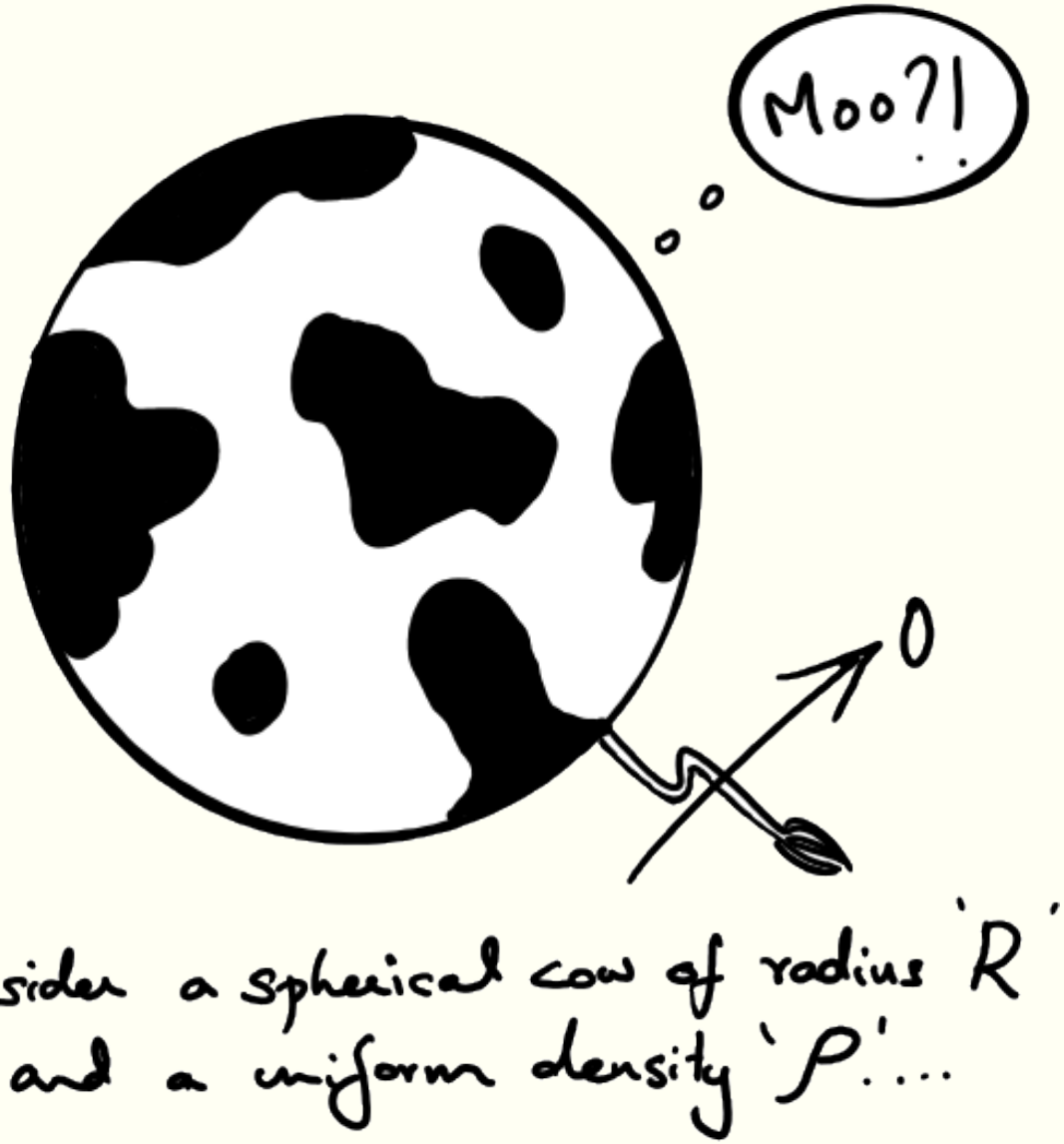
Disney's Frozen: Modeling snow

How to bury Anna under the snow?

How to animate like Disney?

How to achieve the snow simulation?

Modeling the real world: Spherical Cow

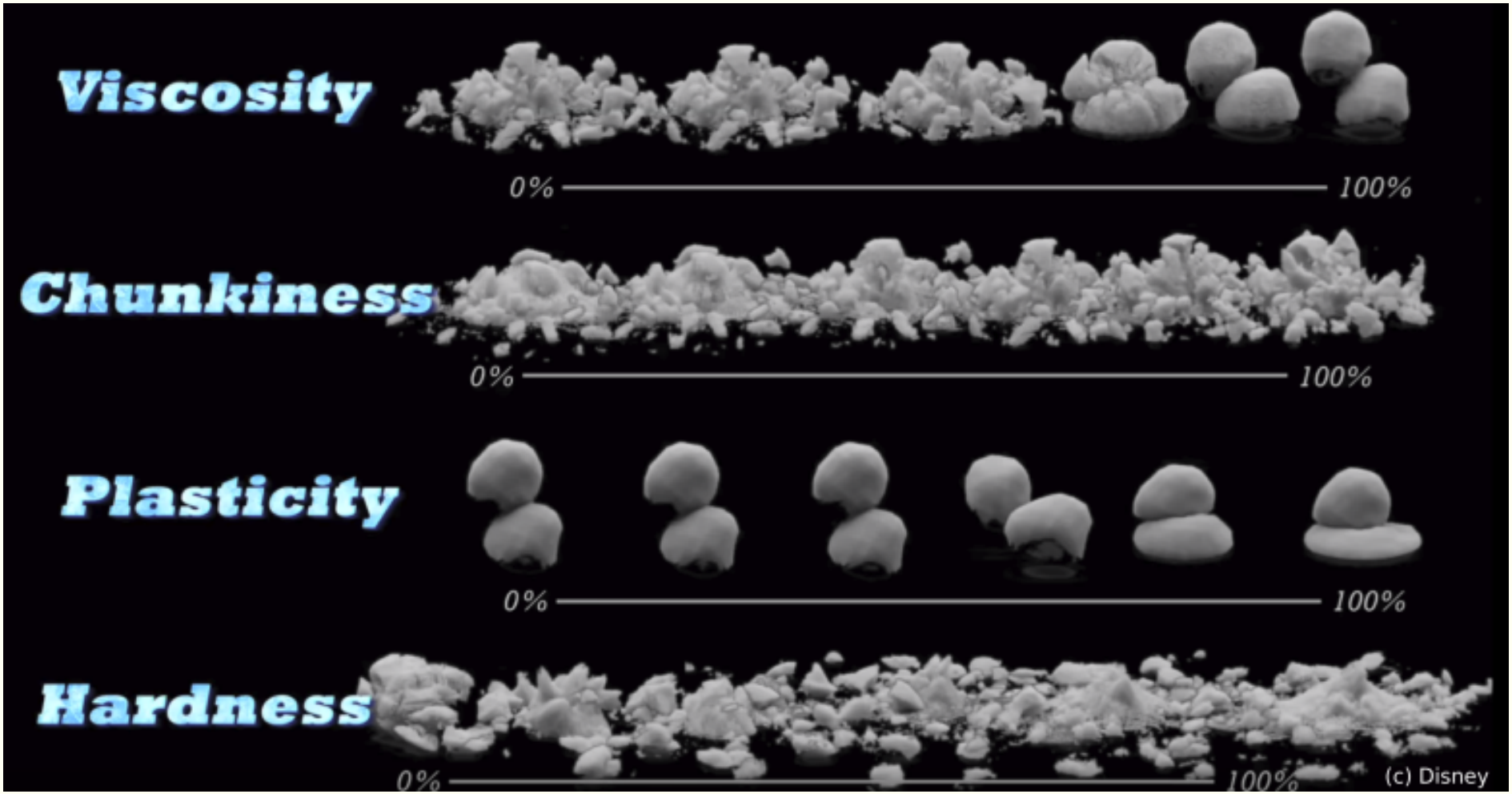


Modeling snow

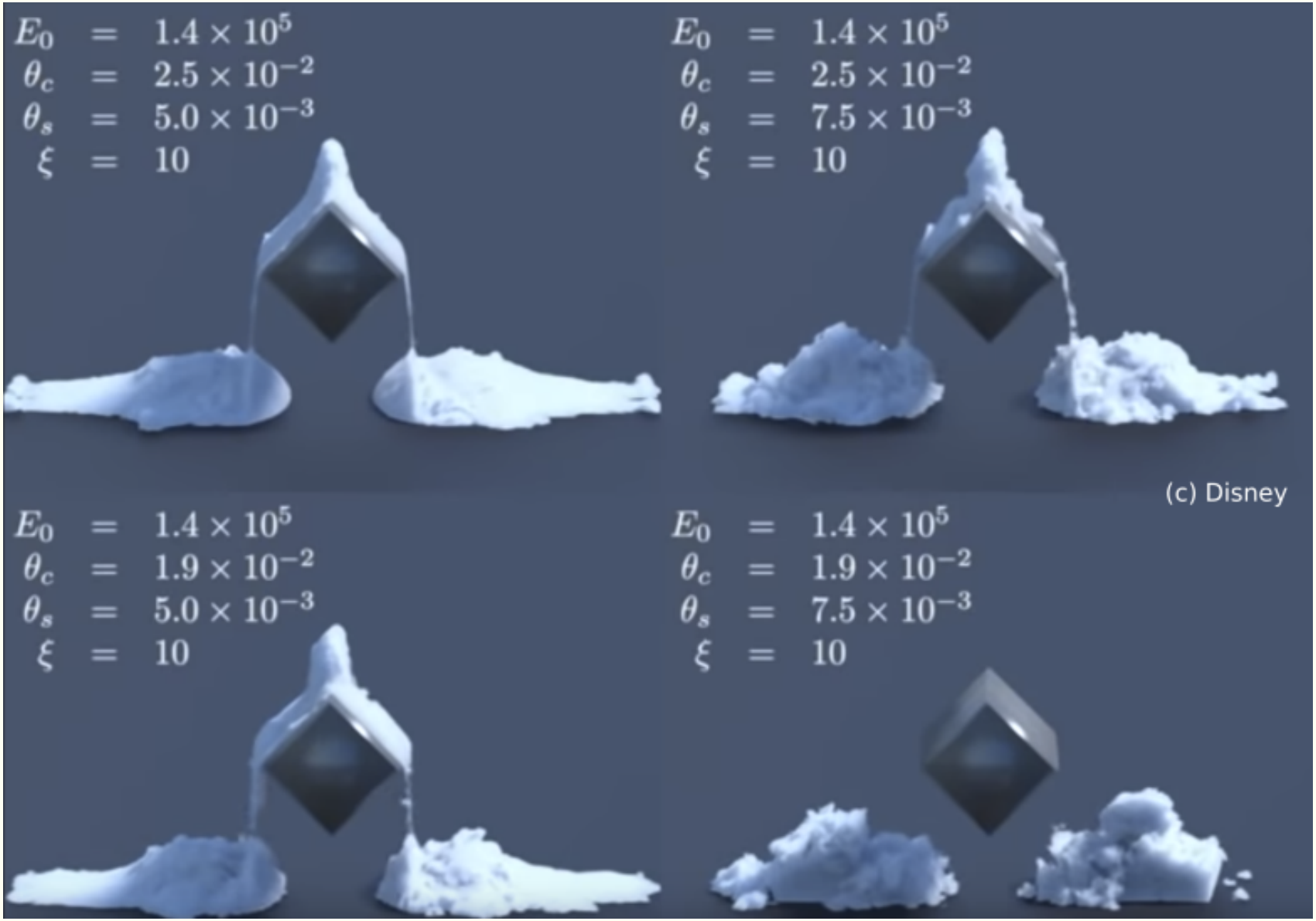
How to animate like Disney: Effect of snow quantity

What type of snow?

Snow properties



Snow material parameters



How to model snow?

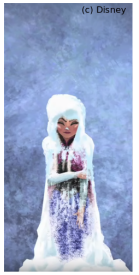
- Representation of snow: Each snow flake is modeled as a particle
- Physics: our model or parameters of how the snow behaves
- Solver: How the snow would move and interact with the objects in the scene.
- Algorithm: A sequence of logical steps required to perform a specific task.
- Implementation and verification of our snow model.

This is a **simulation**.

How to model snow?

- Representation of snow: Each snow flake is modeled as a particle
- Physics: our model or parameters of how the snow behaves
- Solver: How the snow would move and interact with the objects in the scene.
- Algorithm: A sequence of logical steps required to perform a specific task.
- Implementation and verification of our snow model.

This is a **simulation**.



A simulation is an approximate imitation of the operation of a process or system. The first step is to create model, which represents the system's key characteristics, such as its behavior, functions and abstract or physical properties. The model represents the system itself, whereas the simulation represents its operation over time.

There are several numerical techniques that can be used to solve a mathematical problem. They differ in accuracy, length of calulations, and difficulty in programming.

Since numerical solutions are an approximation and the computer program that executes the numerical method might have errors (or bugs), a numerical solution needs to be examined closely.

What does a computer do?

- Fundamentally:
  - performs *calculations* - a billion calculations per second.
  - Remembers results 100s of gigabytes of storage.
- Computers only know and do what you tell them!
- **Fixed program** computer (calculator)
- **Stored program** computer - machine stores and executes instructions

IBM Bluegene/P supercomputer (credit: unknown)

What does a computer do?

- Fundamentally:
  - performs *calculations* - a billion calculations per second.
  - Remembers results 100s of gigabytes of storage.
- Computers only know and do what you tell them!
- **Fixed program** computer (calculator)
- **Stored program** computer - machine stores and executes instructions

IBM Bluegene/P supercomputer (credit: unknown)

A single rack of IBM Bluegene/P in 2007 could do 5Terra flops as fast as the current iPhones X.

UT Austin's Frontera can do 38 PetaFlops. In contrast a human brain can do 1,000 petaflops or 10 quadrillion operations per second (i.e., 1 exaflop or a billion - billion calculations).

Aspects of languages: Primitive constructs

- Turing showed that you can compute anything using **6 primitives**.
- Anything computable in one language is computable in any other programming language.
- *English*: words
- *Programming languages*: numbers, strings, simple operators

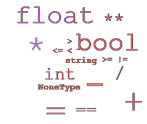
English word cloud (credit: Michael Twardos)

Python word cloud (credit: unknown)

Aspects of languages: Primitive constructs

- Turing showed that you can compute anything using **6 primitives**.
- Anything computable in one language is computable in any other programming language.
- *English*: words
- *Programming languages*: numbers, strings, simple operators

English word cloud (credit: Michael Twardos)



Python word cloud (credit: unknown)

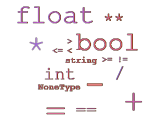
In fact, Alan Turing, who was a really great computer scientist, he showed that you can compute anything using the six primitives. And the six primitives are move left, move right, read, write, scan, and do nothing. Using those six instructions and the piece of tape, he showed that you can compute anything. And using those six instructions, programming languages came about that created a moreconvenient set of primitives. You don't have to program in only these six commands.

CE 311K: Intro to Computer Methods
Aspects of languages
Aspects of languages: Primitive constructs

Aspects of languages: Primitive constructs

- Turing showed that you can compute anything using **6 primitives**.
- Anything computable in one language is computable in any other programming language.
- *English*: words
- *Programming languages*: numbers, strings, simple operators

English word cloud (credit: Michael Twardos)



Python word cloud (credit: unknown)

What are the primary aspects of a language?

- Primitive constructs (objects and operators)
- Syntax
- Static semantics

Aspects of languages: Syntax and Static semantics

Syntax

- *English*
  - "*boy dog cat*": syntactically invalid
  - "*boy hugs cat*": syntactically valid
- *Python*
  - `5 "Hi"`: syntactically invalid
  - `3.2 * 7`: syntactically valid

Static semantics is when syntactically valid strings have meaning:

- English: "*I are hungry*": syntactically valid, but static semantic error
- Python: `5 + "Hi"`: syntactically valid, but static semantic error

Aspects of languages: Semantics

Semantics is the meaning associated with a syntactically correct string of symbols with no static semantic errors:

- *English*: can have many meanings "Flying planes can be dangerous".
- *programming languages*: have only one meaning but may not be what programmer intended

Where things can go wrong

- **syntactic errors**: - common and easily caught
- **static semantic errors**:
  - some languages check for these before running program
  - can cause unpredictable behavior
- no semantic errors **but different meaning than what programmer intended**
  - program crashes, stops running
  - program runs forever
  - program gives an answer but different than expected

Python program

- We'll be using Python 3.x
- a program is a sequence of definitions and commands
  - **definitions** *evaluated*
  - **commands (statements)** instructs Python interpreter (program) to *execute* something
- programs manipulate **data objects**
- objects are
  - **scalar** (cannot be subdivided)
  - *non-scalar* (have internal structure that can be accessed)

- We'll be using Python 3.x
- a program is a sequence of definitions and commands
  - **definitions** *evaluated*
  - **commands (statements)** instructs Python interpreter (program) to *execute* something
- programs manipulate **data objects**
- objects are
  - **scalar** (cannot be subdivided)
  - *non-scalar* (have internal structure that can be accessed)

objects have a **type** that defines the kinds of things programs can do to them:

- Krishna is a human so he can walk, write Python code, speak English, etc.
- Chewbacca is a wookie so he can walk, "mwaaarhrhh", etc.

Scalar objects

- `int`: represent integers, e.g., `5`
- `float`: represent real numbers, e.g., `3.27`
- `bool`: represent Boolean values `True` and `False`
- `NoneType`: special and has one value, `None`
- can use `type()` to see the type of an object

```
In [1]: type(5)
Out[1]: int
In [2]: type(3.0)
Out[2]: float
```

Printing to console

To show the output from code to a user `print` command:

```
In [1]: 3+2
Out[1]: 5    "out": interactive shell

In [2]: print(3+2)
        5    "No out": Shown to user
```

Expressions and operations

- **combine objects and operators** to form *expressions*
- an expression has a value, which has a type
- syntax for a simple expression

  `<object> <operator> <object>`
- Operations on ‘ints‘ and ‘floats‘:
- `i + j`: *sum* (int || float)
- `i - j`: *difference* (int || float)
- `i * j`: *product* (int || float)
- `i / j`: *division* (float)
- `i % j`: *remainder* of `i` divided by `j`
- `i ** j`: `i` to the *power* of `j`

Binding variables and values

- equal sign is an **assignment** of a value to a variable name.

```
pi = 3.14159
pi_approx = 22/7
```

- value stored in computer memory
- an assignment binds name to value
- retrieve value associated with name or variable by invoking the name, by typing `pi`
- Why give names to values of expressions?
  - to reuse names instead of values
  - easier to change code later
- Programming vs math: in programming, you do not "*solve for x*". An assignment expression on the right, evaluated to a value. A variable name defined on the left stores the value(s).

`pi` is the variable

`3.14159` is the value

Binding variables and values

- equal sign is an **assignment** of a value to a variable name.

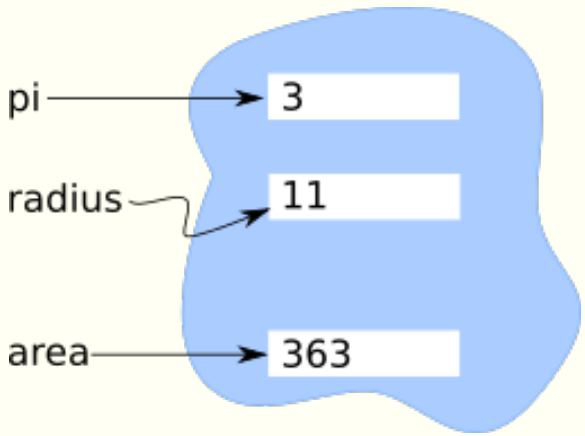
```
pi = 3.14159
pi_approx = 22/7
```

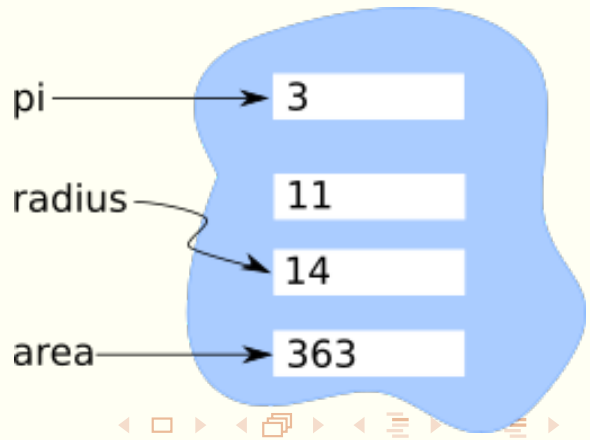
- value stored in computer memory
- an assignment binds name to value
- retrieve value associated with name or variable by invoking the name, by typing `pi`
- Why give names to values of expressions?
  - to reuse names instead of values
  - easier to change code later
- Programming vs math: in programming, you do not "*solve for x*". An assignment expression on the right, evaluated to a value. A variable name defined on the left stores the value(s).

Changing bindings

- can *re-bind* variable names using new assignment statements
- previous value may still stored in memory but lost the handle for it
- value for area does not change until you tell the computer to do the calculation again

```
pi = 3
radius = 11
area = 363
radius = 14
```





Engineering approximations

π: 3.1415926535897 93

e: 2.7182818284590452

Engineers:

Naming matters

? What's wrong with the following code segment?

```
a = 3.14159
b = 11.2
c = a*(b**2)
print(c)
```

```
pi = 3.14159
diameter = 11.2
area = pi*(diameter**2)
print(area)
```

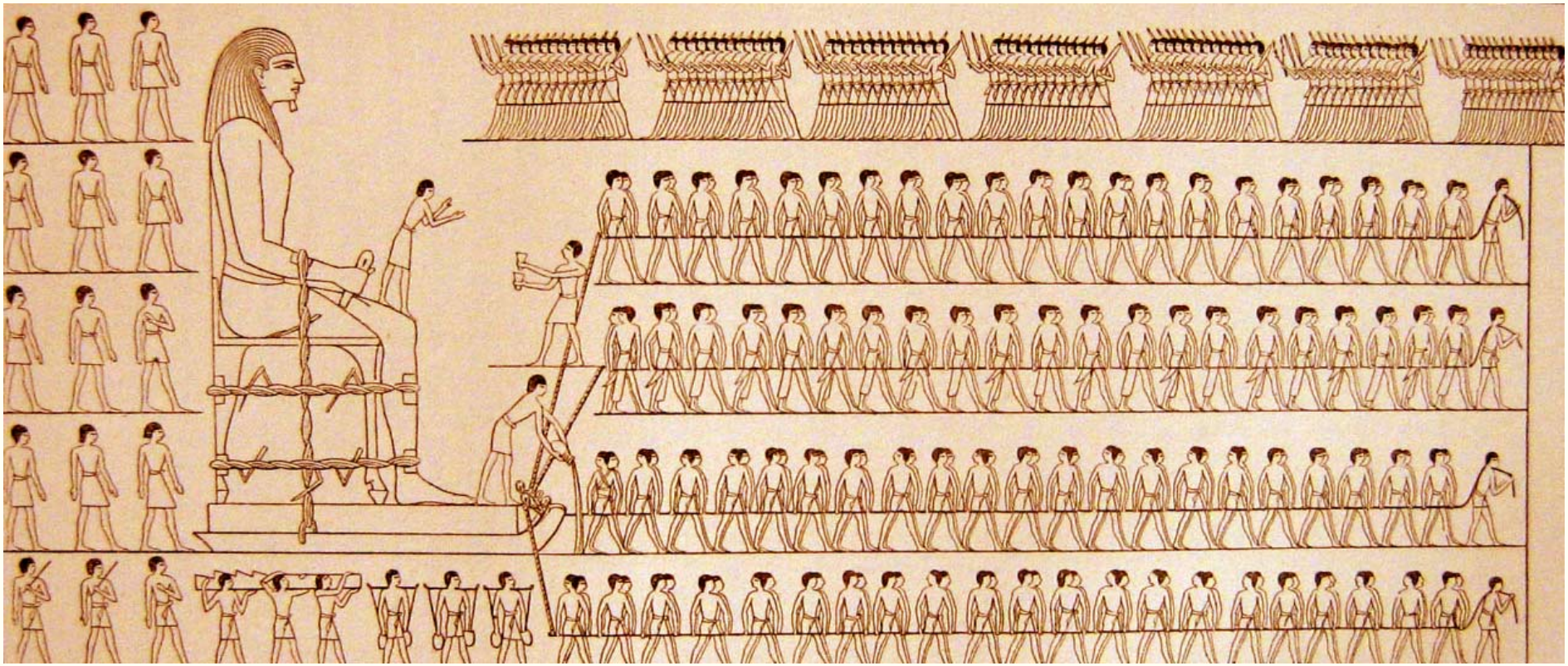
? What's wrong with the following code segment?

```
a = 3.14159
b = 11.2
c = a*(b**2)
print(c)
```

```
pi = 3.14159
diameter = 11.2
area = pi*(diameter**2)
print(area)
```

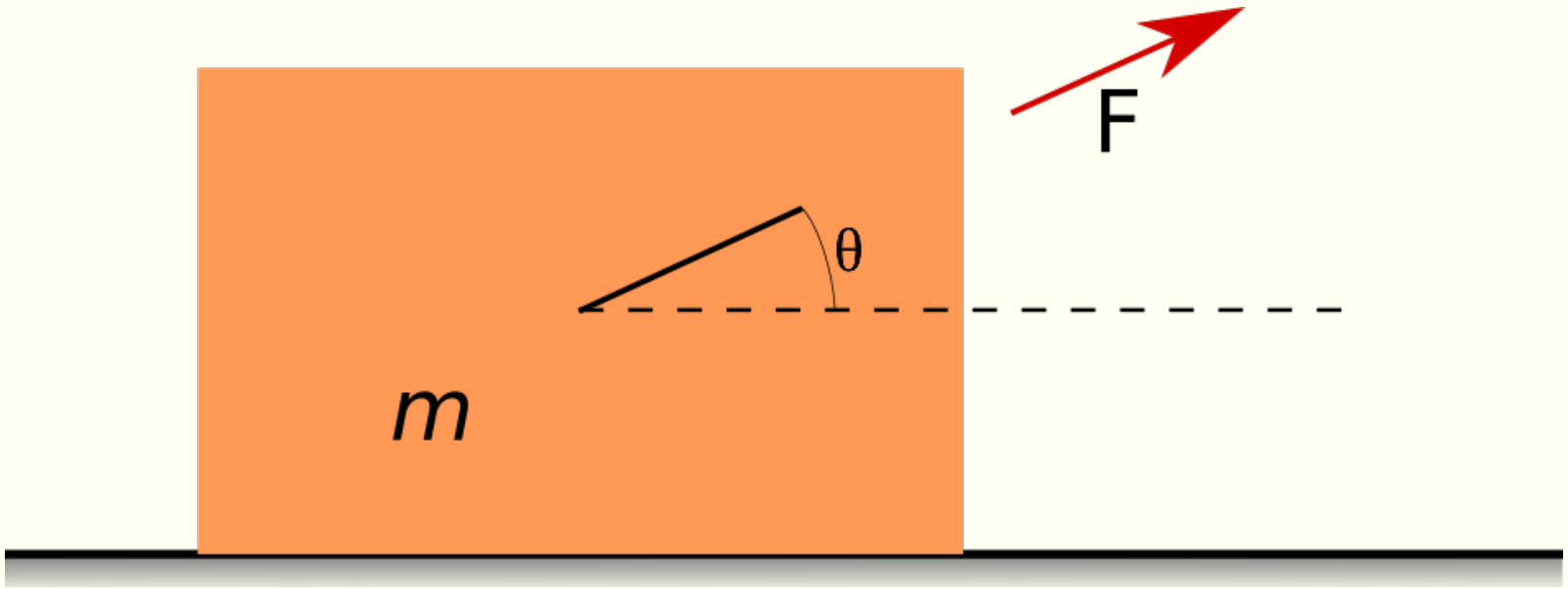
As far as Python is concerned, they are not different. When executed, they will do the same thing. To a human reader, however, they are quite different. When we read the fragment on the left, there is no *a priori* reason to suspect that anything is amiss. However, a quick glance at the code on the right sould prompt us to be suspicious that something is wrong. Either the variable should have been named `radius` rather than `diameter` or `diameter` should have been divided by `2.0` in the area calculation.

What is the optimal angle to pull the statue?



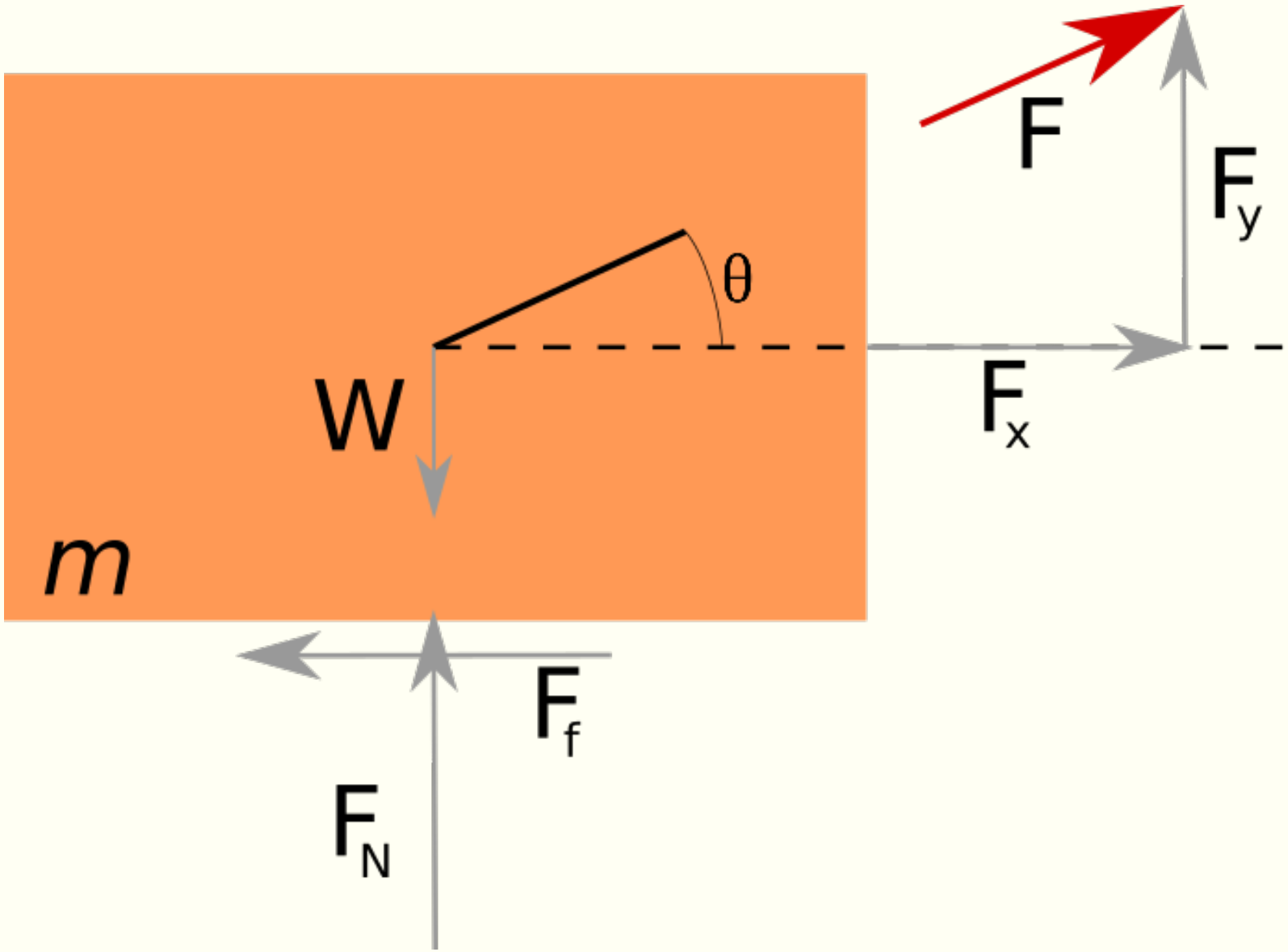
A wall painting from the tomb of Djehutihotep (credit: martinhumanities.com)

Numerical solution of a sliding block: Approximation



What is the optimal angle to pull the block applying the least amount of force?

Numerical solution of a sliding block: Forces



Numerical solution of a sliding block: Forces

$$F_x = F\cos\theta \quad \& \quad F_y = F\sin\theta$$

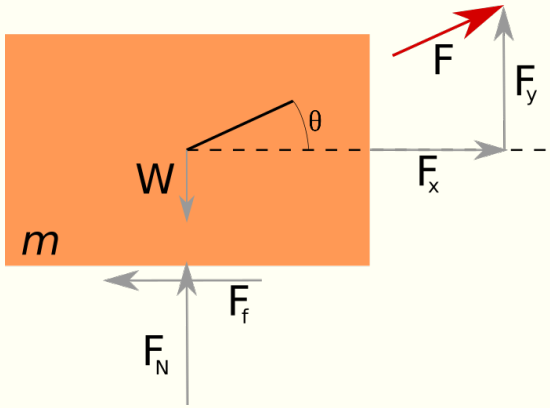
$$F_f = \mu \cdot F_N = \mu \cdot W - \mu F_y = \mu mg - \mu F\sin\theta$$

Vertical forces $\sum F_{vert} \uparrow$: $F_y + F_N - W = 0$

$$F_N = \mu mg - F\sin\theta$$

Horizontal forces $\sum F_{hor} \rightarrow$: $F_x + F_f = 0$

$$F\cos\theta - \mu mg + \mu F\sin\theta = 0$$



$$F = \frac{\mu \cdot mg}{(\cos\theta + \mu\sin\theta)}$$

Numerical solution of a sliding block: Compute force

- Given $W = 25kN(2500\,\text{kg})$, $\theta = 45^o$ and $\mu = 0.75$ (35^o):

$$F = \frac{0.75 \times 25}{\cos(45) + 0.75\sin(45)} = 15.15\,\text{kN}.$$

- Given $F = 17.5kN(2500\,\text{kg})$ and $\mu = 0.75$, what's θ?

$$Try\ \theta = 60^o: \quad F = \frac{0.75 \times 25}{\cos(60) + 0.75\sin(60)} = 16.31\,\text{kN}.$$

$$Try\ \theta = 70^o: \quad F = \frac{0.75 \times 25}{\cos(70) + 0.75\sin(70)} = 17.91\,\text{kN}.$$

$$Try\ \theta = 65^o: \quad F = \frac{0.75 \times 25}{\cos(65) + 0.75\sin(65)} = 17.00\,\text{kN}.$$

$$Try\ \theta = 67,5^o: \quad F = \frac{0.75 \times 25}{\cos(67.5) + 0.75\sin(67.5)} = 17.43\,\text{kN}.$$

This is **bisection method!**

What are the characteristics of a numerical solution?

- A numerical recipe is *a sequence of simple steps*
- *Flow of control* as each step is executed.
- Yields an *approximate* numerical answer (a finite number) for the problem
- These solutions can be very accurate
- Most answers are determined in an iterative approach (numerical method: mathematical / computer-aided technique) until a desired minimum/acceptable accuracy is obtained
- Typically, a finite set of iterations (steps) are used in the numerical method to obtain a solution. A means of determining *when to stop*.

Numerical solution of a sliding block: Friction angles