Introduction to programming

DM536 & DM574

Siddharth Bhaskar

Fall semester 2025

While loops

- `while` *(boolean-expr)* :
  (code)
  ⋮
- This repeatedly executes *code* as long as *boolean-expr* remains true.
- What does the following snippet of code do to the integer variable `n`?

```
while n % 10 == 0:
  n //= 10
```

- Unlike a for-each loop, we don't know in advance how many times the code will execute: less structured, but more powerful.

Primality

- A positive natural number greater than 1 is *prime* if it has no divisors other than 1 and itself.
  - Primes: 2, 3, 5, 7, 11, ...
  - Non-prime composite numbers: 4, 6, 8, 9, 10, 12, ...
- Come up with a simple function

  `is_prime(n: int) -> bool`

  that determines whether a given integer (greater than 1) is prime.
- Observation: if n is composite, then its least divisor must be at most $\sqrt{n}$.

Physical simulation

- *Numerical analysis* and *scientific computation* roughly refer to the idea of modeling physical processes on the computer.
- Problem: model the height of a falling body as a function of time.
- Solution: keep track of the acceleration, velocity, height, and time of the object using the physical laws $a = \frac{dv}{dt}$ and $v = \frac{dh}{dt}$.
  - So $dv = a \cdot dt$ and $dh = v \cdot dt$.
  - Update quantities in small discrete time steps, e.g., $v_{new} = v_{old} + dv$.
- Assume acceleration a is a constant -10 meters per second squared.

Lexicographic ordering

- Problem: given two lists of integers, determine which one is *lexicographically prior:*
- Read the lists in parallel from left-to-right, stopping at the first index at which they differ. The list with the smaller corresponding value is prior.
  - $[1, 2, 3, 4]$ vs. [1,2,4]
  - $[1, 2, 3, 4]$ vs. [1,2,4,3]
  - $[1, 2, 3, 4]$ vs. [1,2,2,4,3]
  - $[5, 2, 4, 6, 6]$ vs. [5,1,100,1000]
- This is basically how you compare words in a dictionary.
- If one list is a *prefix* of another, it is lexicographically prior.
  - [], [5], [5,2], [5,2,4], etc., are all prior to [5,2,4,6,6]

Exponentiation

- Problem: how do you compute e^x given x?
- Idea: use the infinite series

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \dots$$

- Keep summing the series until the tails are too small for python to detect.

Factorization revisited

- Suppose instead of just detecting whether a positive integer is prime or not, we want to *list all of its prime factors.* For example:
  - 12: 2, 2, 3
  - 13: 13
  - 26: 2, 13
  - 99: 3, 3, 11
  - 200: 2, 2, 2, 5, 5
- This is a natural setting to use *nested while loops.*