COMPSCI 590N
Lecture 3: Classes and Representing Numbers

Roy J. Adams

College of Information and Computer Sciences
University of Massachusetts Amherst

Outline

Importing Functions

- **Modules** are Python packages, usually a collection of functions and variable, that can be used in other programs.

Importing Functions

- **Modules** are Python packages, usually a collection of functions and variable, that can be used in other programs.
- Load modules into your code using an **import** statement.

Importing Functions

- **Modules** are Python packages, usually a collection of functions and variable, that can be used in other programs.
- Load modules into your code using an **import** statement.

Importing Functions

- **Modules** are Python packages, usually a collection of functions and variable, that can be used in other programs.
- Load modules into your code using an **import** statement.

```
import <module_name> # imports a module
```

Useful Built-in Modules

Python has a number of very useful built-in modules:

- **string:** operations on strings.

Useful Built-in Modules

Python has a number of very useful built-in modules:

- **string:** operations on strings.
- **math:** standard math functions such as: log, exp, sine, etc.

Useful Built-in Modules

Python has a number of very useful built-in modules:

- **string:** operations on strings.
- **math:** standard math functions such as: log, exp, sine, etc.
- **itertools:** functions for manipulating sequences such as: combinations, permutations, cross product, etc.

Useful Built-in Modules

Python has a number of very useful built-in modules:

- **string:** operations on strings.
- **math:** standard math functions such as: log, exp, sine, etc.
- **itertools:** functions for manipulating sequences such as: combinations, permutations, cross product, etc.

Useful Built-in Modules

Python has a number of very useful built-in modules:

- **string:** operations on strings.
- **math:** standard math functions such as: log, exp, sine, etc.
- **itertools:** functions for manipulating sequences such as: combinations, permutations, cross product, etc.

DEMO

Object Oriented Programming

DEMO

Classes

Create new object types in Python by defining a **class**.

Classes

Create new object types in Python by defining a **class**.

```
class <class_name>:
    def __init__(self,<args>):
        self.<member_variable> = <expression>
        <body>

    def <member_function>(self,<args>):
        <body>
```

What are objects?

- Objects, like functions, are a tool for organizing programs.

What are objects?

- Objects, like functions, are a tool for organizing programs.
- An object consists of:

What are objects?

- Objects, like functions, are a tool for organizing programs.
- An object consists of:
  1. A collection of related information.

What are objects?

- Objects, like functions, are a tool for organizing programs.
- An object consists of:
  1. A collection of related information.
  2. A set of operations to manipulate and access that information.

What are objects?

- Objects, like functions, are a tool for organizing programs.
- An object consists of:
  1. A collection of related information.
  2. A set of operations to manipulate and access that information.
- The information is stored as *instance variables*.

What are objects?

- Objects, like functions, are a tool for organizing programs.
- An object consists of:
  1. A collection of related information.
  2. A set of operations to manipulate and access that information.
- The information is stored as *instance variables*.
- The operations are called *methods*.

What are objects?

- Objects, like functions, are a tool for organizing programs.
- An object consists of:
  1. A collection of related information.
  2. A set of operations to manipulate and access that information.
- The information is stored as *instance variables*.
- The operations are called *methods*.
- Collectively the methods and instance variables are called *attributes*.

What are objects?

- Objects, like functions, are a tool for organizing programs.
- An object consists of:
  1. A collection of related information.
  2. A set of operations to manipulate and access that information.
- The information is stored as *instance variables*.
- The operations are called *methods*.
- Collectively the methods and instance variables are called *attributes*.

What are objects?

- Objects, like functions, are a tool for organizing programs.
- An object consists of:
  1. A collection of related information.
  2. A set of operations to manipulate and access that information.
- The information is stored as *instance variables*.
- The operations are called *methods*.
- Collectively the methods and instance variables are called *attributes*.

Dog Example:

- **Instance variables**:

What are objects?

- Objects, like functions, are a tool for organizing programs.
- An object consists of:
  1. A collection of related information.
  2. A set of operations to manipulate and access that information.
- The information is stored as *instance variables*.
- The operations are called *methods*.
- Collectively the methods and instance variables are called *attributes*.

Dog Example:

- **Instance variables**:
  - `name` and `tricks`

What are objects?

- Objects, like functions, are a tool for organizing programs.
- An object consists of:
  1. A collection of related information.
  2. A set of operations to manipulate and access that information.
- The information is stored as *instance variables*.
- The operations are called *methods*.
- Collectively the methods and instance variables are called *attributes*.

Dog Example:

- **Instance variables**:
  - `name` and `tricks`
- **Methods:**

What are objects?

- Objects, like functions, are a tool for organizing programs.
- An object consists of:
    1. A collection of related information.
    2. A set of operations to manipulate and access that information.
- The information is stored as *instance variables*.
- The operations are called *methods*.
- Collectively the methods and instance variables are called *attributes*.

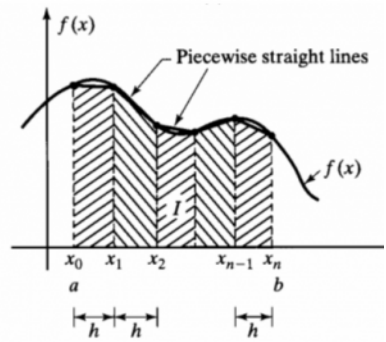
Dog Example:

- **Instance variables**:
    - `name` and `tricks`
- **Methods:**
    - `bark`, `teach_trick`, and `do_trick`

Outline

Numerical Computing

Numerical computing is the approximation of continuous values and functions on a computer with finite precision.



Representing Numbers

Before we can approximate functions, we need to represent numbers:

Representing Numbers

Before we can approximate functions, we need to represent numbers:

- The set of integers is countably infinite.

Representing Numbers

Before we can approximate functions, we need to represent numbers:

- The set of integers is countably infinite.
- The set of real numbers is continuous and uncountably infinite.

Representing Numbers

Before we can approximate functions, we need to represent numbers:

- The set of integers is countably infinite.
- The set of real numbers is continuous and uncountably infinite.
- But the set of numbers representable by a computer is finite...

Integer Representation

Given a fixed number of bits (usually 32 or 64), we can create the following mapping:

Integer Representation

Given a fixed number of bits (usually 32 or 64), we can create the following mapping:

integral number	bit representation
0	00000000
1	00000001
2	00000010
3	00000011
4	00000100
⋮	⋮
254	11111110
255	11111111

Integer Representation

Given a fixed number of bits (usually 32 or 64), we can create the following mapping:

integral number	bit representation
0	00000000
1	00000001
2	00000010
3	00000011
4	00000100
$\vdots$	$\vdots$
254	11111110
255	11111111

- Shift the mapping to store negative numbers: $0000000_2 = -125$ and $11111111_2 = 126$.

Integer Representation

Binary Integers

Given n bits, $b_1, ..., b_n \in \{0, 1\}$, we map binary values to integer values as follows:

$$(b_n b_{n-1} ... b_2 b_1 b_0)_2 = \sum_{i=0}^{n} b_i 2^i = x$$

Integer Representation

Binary Integers

Given n bits, $b_1, \ldots, b_n \in \{0, 1\}$, we map binary values to integer values as follows:

$$(b_n b_{n-1} \ldots b_2 b_1 b_0)_2 = \sum_{i=0}^{n} b_i 2^i = x$$

For example:

$$0101_2 = 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 5$$

Real Representations

Fixed Point

Represent real numbers as integers that are scaled by a fixed scaling factor, d. For example: Let $d = 1000$, then

$$1.23 = \frac{12300}{d}$$

Real Representations

Fixed Point

Represent real numbers as integers that are scaled by a fixed scaling factor, d. For example: Let $d = 1000$, then

$$1.23 = \frac{12300}{d}$$

- This is equivalent to shifting the decimal.

Real Representations

Fixed Point

Represent real numbers as integers that are scaled by a fixed scaling factor, d. For example: Let $d = 1000$, then

$$1.23 = \frac{12300}{d}$$

- This is equivalent to shifting the decimal.
- d is usually a multiple of 2.

Real Representations

Fixed Point

Represent real numbers as integers that are scaled by a fixed scaling factor, d. For example: Let $d = 1000$, then

$$1.23 = \frac{12300}{d}$$

- This is equivalent to shifting the decimal.
- d is usually a multiple of 2.
- Restricted by the range of representable integers.

Real Representations

Fixed Point

Represent real numbers as integers that are scaled by a fixed scaling factor, d. For example: Let $d = 1000$, then

$$1.23 = \frac{12300}{d}$$

- This is equivalent to shifting the decimal.
- d is usually a multiple of 2.
- Restricted by the range of representable integers.
- **Observation:** Between 0 and 1, we would usually like a finer discretization, but between 1000 and 1001, we may be ok with a rough discretization.

Real Representations

Floating Point

Rewrite a number, x, in scientific notation $x = a \cdot 10^b$. Then, x can be stored by storing a as a fixed point and b as an integer. Floating point numbers use a fixed number of digits for a, known as the *mantissa*, and b, known as the *exponent*:

Real Representations

Floating Point

Rewrite a number, x, in scientific notation $x = a \cdot 10^b$. Then, x can be stored by storing a as a fixed point and b as an integer. Floating point numbers use a fixed number of digits for a, known as the *mantissa*, and b, known as the *exponent*:

x	a	b
123.456	1.23456	2
100000	1.00000	5
0.00025	2.50000	-4

Real Representations

Floating Point

Rewrite a number, x, in scientific notation $x = a \cdot 10^b$. Then, x can be stored by storing a as a fixed point and b as an integer. Floating point numbers use a fixed number of digits for a, known as the *mantissa*, and b, known as the *exponent*:

x	a	b
123.456	1.23456	2
100000	1.00000	5
0.00025	2.50000	-4

- Floating point numbers naturally have finer granularity nearer zero.

Floating Point Limitations

- Typically, 54 bits are used for the mantissa and 10 bits are used for the exponent.

Floating Point Limitations

- Typically, 54 bits are used for the mantissa and 10 bits are used for the exponent.
- This results in:

Floating Point Limitations

- Typically, 54 bits are used for the mantissa and 10 bits are used for the exponent.
- This results in:
  - The largest possible float is $\approx 10^{308}$.

Floating Point Limitations

- Typically, 54 bits are used for the mantissa and 10 bits are used for the exponent.
- This results in:
  - The largest possible float is $\approx 10^{308}$.
  - The smallest possible float is $\approx 10^{-308}$.

Floating Point Limitations

- Typically, 54 bits are used for the mantissa and 10 bits are used for the exponent.
- This results in:
  - The largest possible float is $\approx 10^{308}$.
  - The smallest possible float is $\approx 10^{-308}$.
  - The distance between 1.0 and the next largest number is $\approx 10^{-16}$.

Floating Point Limitations

- Typically, 54 bits are used for the mantissa and 10 bits are used for the exponent.
- This results in:
  - The largest possible float is $\approx 10^{308}$.
  - The smallest possible float is $\approx 10^{-308}$.
  - The distance between 1.0 and the next largest number is $\approx 10^{-16}$.
- Because we can only represent finite numbers, we must rely on rounding and approximation which can lead to errors if you are not careful.

Overflow/Underflow

Overflow/Underflow

Overflow

Overflow occurs when an expression results in a number that is too **large** to be represented.

Overflow/Underflow

Overflow

Overflow occurs when an expression results in a number that is too **large** to be represented.

Underflow

Underflow occurs when an expression results in a number that is too **small** to be represented.

Overflow/Underflow

Overflow

Overflow occurs when an expression results in a number that is too **large** to be represented.

Underflow

Underflow occurs when an expression results in a number that is too **small** to be represented.

Demo

Representability

- Not all decimal numbers are representable using a binary, floating point representation.

Representability

- Not all decimal numbers are representable using a binary, floating point representation.
- Example: 0.1

Representability

- Not all decimal numbers are representable using a binary, floating point representation.
- Example: 0.1
- The set of representable numbers is not closed under standard arithmetic. That is, adding, subtracting, multiplying, or dividing representable numbers may result in an unrepresentable number.

Representability

- Not all decimal numbers are representable using a binary, floating point representation.
- Example: 0.1
- The set of representable numbers is not closed under standard arithmetic. That is, adding, subtracting, multiplying, or dividing representable numbers may result in an unrepresentable number.
- Example: 1.0/10.0 = 0.1

Representability

- Not all decimal numbers are representable using a binary, floating point representation.
- Example: 0.1
- The set of representable numbers is not closed under standard arithmetic. That is, adding, subtracting, multiplying, or dividing representable numbers may result in an unrepresentable number.
- Example: $1.0/10.0 = 0.1$

Representability

- Not all decimal numbers are representable using a binary, floating point representation.
- Example: 0.1
- The set of representable numbers is not closed under standard arithmetic. That is, adding, subtracting, multiplying, or dividing representable numbers may result in an unrepresentable number.
- Example: 1.0/10.0 = 0.1

Demo

Working with floating point numbers

- Scale your variables or work in log space to avoid overflow and underflow.

Working with floating point numbers

- Scale your variables or work in log space to avoid overflow and underflow.
- Avoid using == to compare floating point numbers. Instead compare with a tolerance: $|x - y| < \epsilon$

Working with floating point numbers

- Scale your variables or work in log space to avoid overflow and underflow.
- Avoid using == to compare floating point numbers. Instead compare with a tolerance: $|x - y| < \epsilon$

Working with floating point numbers

- Scale your variables or work in log space to avoid overflow and underflow.
- Avoid using == to compare floating point numbers. Instead compare with a tolerance: $|x - y| < \epsilon$

Demo

Rounding in Action

Can a 10^{-16} make a difference?

Rounding in Action

Can a 10^{-16} make a difference?

Patriot Missile Failure

- Missile defense system failed to target an incoming missile due to a rounding error.

Rounding in Action

Can a 10^{-16} make a difference?

Patriot Missile Failure

- Missile defense system failed to target an incoming missile due to a rounding error.
- The error was caused by counting time in tenths of seconds (+= 0.1).

Rounding in Action

Can a 10^{-16} make a difference?

Patriot Missile Failure

- Missile defense system failed to target an incoming missile due to a rounding error.
- The error was caused by counting time in tenths of seconds (+= 0.1).
- As in our demo, this resulted in accumulating errors because 0.1 is not representable.

Rounding in Action

Can a 10^{-16} make a difference?

Patriot Missile Failure

- Missile defense system failed to target an incoming missile due to a rounding error.
- The error was caused by counting time in tenths of seconds (+= 0.1).
- As in our demo, this resulted in accumulating errors because 0.1 is not representable.
- Incorrect timestamps were then used to incorrectly calculate distance and speed of an incoming missile.