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 (logical) error. Your program runs without error messages, but does not do what it is supposed to do.

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.

Formal and natural languages

- Natural languages: languages that people speak. They were not designed by people; they evolved naturally. (ambiguous: we use contextual clues and other info to deal with this. Full of metaphors and idioms)

- Formal languages: designed by people for specific applications. Have strict syntax rules.(tokens & struct.)

  3+3=6 vs 3$=

  H_2O vs $_2Zz$

(not ambiguous : each stmt has exactly one meaning regardless of context.)

- Programming languages are formal languages that have been designed to express computations.

What is a programing language?

High-level programming languages: Java, Python, C, C++, Perl

- Enormous advantages: much easier (less time, shorter, easier to read), more likely to be correct, portable.
- Small disadvantage: have to be translated before running (compiling or interpreting)
- Thus, almost all programs written in them

Low-level programming languages: machine and assembly

- Big disadvantages: only run on one kind of computer. Have to be rewritten to run on another
- Thus only used for a few special applications

Interpreting / Compiling

There are 2 ways to translate a program written in high level language (also called source code):

Interpreting: An interpreter is a program that reads a high-level program and does what it says.

Compiling: A complier is a program that reads a high-level program and translates it all at once, before running any of the commands. It complies a program first and then runs the complied code later. The complied code is called executable.

Algebraic expressions

The Python interactive shell can be used to evaluate algebraic expressions

`14//3` is the quotient when 14 is divided by 3 and `14%3` is the remainder

`2**3` is 2 to the 3rd power

`abs()`, `min()`, and `max()` are functions

- `abs()` takes a number as input and returns its absolute value
- `min()` (resp., `max()`) take an arbitrary number of inputs and return the "smallest" (resp., "largest") among them

```
>>> 2 + 3
5
>>> 7 - 5
2
>>> 2*(3+1)
8
>>> 5/2
2.5
>>> 5//2
2
>>> 14//3
4
>>> 14%3
2
>>> 2**3
8
```

Boolean expressions

In addition to algebraic expressions, Python can evaluate Boolean expressions

- Boolean expressions evaluate to `True` or `False`
- Boolean expressions often involve comparison operators
  <, >, ==, !=, <=, and >=

```
>>> 2 < 3
True
>>> 2 > 3
False
>>> 2 == 3
False
>>> 2 != 3
True
>>> 2 <= 3
True
>>> 2 >= 3
False
>>> 2+4 == 2*(9/3)
True
```

In a an expression containing algebraic and comparison operators:

- Algebraic operators are evaluated first
- Comparison operators are evaluated next

Boolean operators

In addition to algebraic expressions, Python can evaluate Boolean expressions

- Boolean expressions evaluate to `True` or `False`
- Boolean expressions may include Boolean operators `and`, `or`, and `not`

```
>>> 2<3 and 3<4
True
>>> 4==5 and 3<4
False
>>> False and True
False
>>> True and True
True
>>> 4==5 or 3<4
True
>>> False or True
True
>>> False or False
False
>>> not(3<4)
False
>>> not(True)
False
>>> not(False)
True
>>> 4+1==5 or 4-1<4
True
```

In a an expression containing algebraic, comparison, and Boolean operators:

- Algebraic operators are evaluated first
- Comparison operators are evaluated next
- Boolean operators are evaluated last

Representation of numbers in Python

Integers can be represented exactly. There is no limits to their size (other than the size of you memory).

Floating points are approximation of real numbers. Python uses a double-precision standard format (IEEE 754) that can represent real numbers in a range of 10^{-308} to 10^{308} with 16 to 17 digits of precision.

Operations : -, **, *, /, %, //, +, -

<, <=, ==, !=, >, >=

== should be avoided in floats

Number-type operators

Table 2.4 Number-type operators. Listed are the operators that can be used on number objects (e.g., `bool`, `int`, `float`). If one of the operands is a `float`, the result is always a `float` value; otherwise, the result is an `int` value, except for the division (`/`) operator, which always gives a `float` value.

Operation	Description	Type (if `x` and `y` are integers)
`x + y`	Sum	Integer
`x - y`	Difference	Integer
`x * y`	Product	Integer
`x / y`	Division	Float
`x // y`	Integer division	Integer
`x % y`	Remainder of `x // y`	Integer
`-x`	Negative `x`	Integer
`abs(x)`	Absolute value of `x`	Integer
`x**y`	`x` to the power `y`	Integer

This table is from “Introduction to Computing Using Python” by Lj. Perkovic

Operator Precedence

Operator	Description
`[expressions...]`	List definition
`x[], x[index:index]`	Indexing operator
`**`	Exponentiation
`+x, -x`	Positive, negative signs
`*, /, //, %`	Product, division, integer division, remainder
`+, -`	Addition, subtraction
`in, not in, <, <=, >, >=, <>, !=, ==`	Comparisons, including membership and identity tests
`not x`	Boolean NOT
`and`	Boolean AND
`or`	Boolean OR

Table 2.6 Operator precedence. The operators are listed in order of precedence from highest on top to lowest at the bottom; operators in the same row have the same precedence. Higher-precedence operations are performed first, and equal precedence operations are performed in left-to-right order.

This table is from "Introduction to Computing Using Python" by Lj. Perkovic

Variables and assignments

Just as in algebra, a value can be assigned to a variable, such as x

When variable x appears inside an expression, it evaluates to its assigned value

A variable (name) does not exist until it is assigned

The assignment statement has the format

```
<variable> = <expression>
```

<expression> is evaluated first, and the resulting value is assigned to variable <variable>

```
>>> x = 3
>>> x
3
>>> 4*x
16
>>> y
Traceback (most recent call
last):
  File "<pyshell#59>", line
1, in <module>
    y
NameError: name 'y' is not
defined
>>> y = 4*x
>>> y
16.0
```

Naming rules

(Variable) names can contain these characters:

- a through z
- A through Z
- the underscore character _
- digits 0 through 9

Names cannot start with a digit though

For a multiple-word name, use

- either the underscore as the delimiter
- or *camelCase* capitalization

Short and meaningful names are ideal

```
>>> My_x2 = 21
>>> My_x2
21
>>> 2x = 22
SyntaxError: invalid syntax
>>> new_temp = 23
>>> newTemp = 23
>>> counter = 0
>>> temp = 1
>>> price = 2
>>> age = 3
```