

Chapter_1_The_way_of_the_program

March 6, 2024

Python is considered an interpreted language because Python programs are executed by an interpreter. **There are two ways to use the interpreter: interactive mode and script mode. In interactive mode, you type Python programs and the interpreter displays the result.**

```
[1]: # checking the python version
import sys
print(sys.version)
```

3.9.18 (main, Sep 11 2023, 14:09:26) [MSC v.1916 64 bit (AMD64)]

The chevron, `»>`, is the prompt the interpreter uses to indicate that it is ready

Working in interactive mode is convenient for testing small pieces of code because you can type and execute them immediately

Programming is error-prone. For whimsical reasons, programming errors are called bugs and the process of tracking them down is called debugging

Three kinds of errors can occur in a program: syntax errors, runtime errors, and semantic errors. It is useful to distinguish between them in order to track them down more quickly.

Syntax refers to the structure of a program and the rules about that structure. For example, parentheses have to come in matching pairs, so `(1 + 2)` is legal, but `8)` is a syntax error.

Python is not so forgiving. If there is a single syntax error anywhere in your program, Python will display an error message and quit, and you will not be able to run your program.

The second type of error is a runtime error, so called because the error does not appear until after the program has started running. These errors are also called exceptions because they usually indicate that something exceptional (and bad) has happened. Runtime errors are rare in the simple programs you will see in the first few chapters, so it might be a while before you encounter one.

The third type of error is the semantic error. If there is a semantic error in your program, it will run successfully in the sense that the computer will not generate any error messages, but it will not do the right thing. It will do something else. Specifically, it will do what you told it to do. The problem is that the program you wrote is not the program you wanted to write. The meaning of the program (its semantics) is wrong. Identifying semantic errors can be tricky because it requires you to work backward by looking at the output of the program and trying to figure out what it is doing.

1.3.4 Experimental debugging

One of the most important skills you will acquire is debugging. Although it can be frustrating, debugging is one of the most intellectually rich, challenging, and interesting parts of programming.

Debugging is also like an experimental science. Once you have an idea about what is going wrong, you modify your program and try again. If your hypothesis was correct, then you can predict the result of the modification, and you take a step closer to a working program. If your hypothesis was wrong, you have to come up with a new one.

1.4 Formal and natural languages

Natural languages are the languages people speak, such as English, Spanish, and French. They were not designed by people (although people try to impose some order on them); they evolved naturally.

Formal languages are languages that are designed by people for specific applications. For example, the notation that mathematicians use is a formal language that is particularly good at denoting relationships among numbers and symbols. Chemists use a formal language to represent the chemical structure of molecules. And most importantly: Programming languages are formal languages that have been designed to express computations. Formal languages tend to have strict rules about syntax. For example, $3 + 3 = 6$ is a syntactically correct mathematical statement, but $3+ = 3\$6$ is not. H_2O is a syntactically correct chemical formula, but $2Zz$ is not

The second type of syntax rule pertains to the structure of a statement; that is, the way the tokens are arranged. The statement $3+ = 3$ is illegal because even though $+$ and $=$ are legal tokens, you can't have one right after the other. Similarly, in a chemical formula the subscript comes after the element name, not before.

When you read a sentence in English or a statement in a formal language, you have to figure out what the structure of the sentence is (although in a natural language you do this subconsciously). This process is called parsing.

For example, when you hear the sentence, "The penny dropped," you understand that "the penny" is the subject and "dropped" is the predicate. Once you have parsed a sentence, you can figure out what it means, or the semantics of the sentence. Assuming that you know what a penny is and what it means to drop, you will understand the general implication of this sentence.

ambiguity: Natural languages are full of ambiguity, which people deal with by using contextual clues and other information. Formal languages are designed to be nearly or completely unambiguous, which means that any statement has exactly one meaning, regardless of context. redundancy: In order to make up for ambiguity and reduce misunderstandings, natural languages employ lots of redundancy. As a result, they are often verbose. Formal languages are less redundant and more concise.

[]: