Python Grammar

Python is a programming language, and like all programming languages it is described by a formal grammar. This allows the Python interpreter to unambiguously determine the meaning of your Python code.

This document is **not** a full, formal definition of Python's grammar, but just the subset of the grammar that we've used in this course. If you're interested in learning more about these, take a look at Context Free Grammars. We've tried to use examples rather than formal definitions to make it easier to learn.

Expressions

Rules are written as `rule-name = different patterns separated by |` (i.e. read `|` as "or").

```
expression  = number
            | arithmetic
            | string
            | function-call
            | variable
            | comparison

number      = any integer or real number
arithmetic  = expression using operators (+, -, *, /, etc)
string      = "text between double quote marks"
variable    = an identifier, not surrounded by quote marks
comaparison = expression using comparison operators (==, !=, <, >, <=, >=) and
              logical connectives (and, or, not)
```

Examples:

- `42`
- `"Hello, world!"`
- `42 * 42`
- `"Hello, " + "world!"`
- `my_variable_name`
- `x == 10`

Identifiers

Identifiers are single words that can be used for the names of variables or functions in Python. In general, we're always going to use lowercase letters for our identifier names, but so long as they start with a letter you can also use upper/lowercase letters, underscores, and digits.

Examples:

- `hello`
- `name123`
- `very_long_identifier_name`

Function calls

```
function-call = identifier(expression, expression, ...)
```

A function can possible return a value, and take any number of arguments, which are expressions. The only mandatory parts are the function name and the brackets.

Examples:

- `print("Hello, world\n")`
- `input()`

Statements

```
statement         = function-call
                  | if-statement
                  | if-else-statement
                  | assignment

if-statement      = if conditional:
                        statement(s) indented by a tab

if-else-statement = if conditional:
                        statement(s) indented by a tab
                    else:
                        statement(s) indented by a tab

assignment = identifier = expression
```

These are all the kinds of statements we've seen so far. This expresses that the `else` branch of an `if` statement is optional, and shows that an `if` statement can contain an `if` statement, or another kind of statement.

Examples:

- `print("Hello, world!")` (function call)
- If statement:

```
python   if x > 10:       print("x > 10")
```

Programs

```
program = statement
          program
        | empty
```

This is a recursive definition, so a program can either be empty (Python will quite happily run an empty file and do nothing), or it can be a statement, followed by a newline, followed by a program. Essentially, **a program is a list of statements**.