CIS 231

Python – Ch. 2

Variables, expressions, and statements

2.1 Assignment statements

- Used to (re)place, or *assign*, values to variables
  - If this is the first mention of the variable, the variable is created (also known as *declaring* a variable) and then assigned the value
- The = operator designates an assignment
- If the variable already exists, it now contains the assigned value; the previous value is discarded

2.2 Variable names and keywords

- Variable names:
  - Arbitrary length
  - Must consist of letters (upper/lower), digits, and/or underscore (_)
  - Must start with a letter, lower case strongly preferred
  - Should be meaningful and not ambiguous
- Keywords (*reserved words*) are set aside by Python for its own use – these cannot be used as variable names
  - List on p. 10

2.3 Expressions and statements

- An *expression* is a combination of values, variables, and/or operators
  - A value or variable alone can be an expression
- An operator and its operand(s) is commonly referred to as an *operation*
- A *statement* is a unit of code that the Python interpreter can convert and execute
  - It may include arithmetic expressions

2.4 Interactive/script modes

- Interactive mode
  - Typing right into the console
  - Ad hoc, but remembers variables/values until a restart
- Script mode
  - Allows for saving/reusing statements in a `.py` script file
  - Can be executed in IDE or on command line

2.5 Order of operations

- Also known as operator precedence
- The process that an arithmetic expression is broken down and executed
  - Not just left-to-right
  - Uses PEMDAS to group operations into levels of precedence – one level executed at a time
  - Operations of same precedence are done L-to-R
  - **P**arentheses, **E**xponentiation, **M**ultiplication and **D**ivision, **A**ddition and **S**ubtraction

2.6 String operations

- Uses some of the same operators as numbers, but they have different meaning
- `+` is the *concatenation* operator – appends one string onto the end of another
  - E.g. “`go “ + “away`” produces “`go away`”
- `*` performs string repetition
  - E.g. ‘`Spam`’`*3` produces ‘`SpamSpamSpam`’

2.7 Comments

- Notes that you add to your program to provide additional info but are not code
- Will not be interpreted by Python, therefore not subject to any syntax rules
- Start the line with # to indicate a comment:
  - `# terribly important but not code`
  - `# Python will ignore these lines`
- Can use quotes for multi-line comments

2.8 Debugging

- The process of finding and correcting errors in your programs
- Error types:
  - *Syntax errors* – code could not be *parsed* because it didn't conform to the language's *syntax*
  - *Runtime errors* (aka *exceptions*) – syntax was ok but an unusual occurrence – e.g. divide by 0
  - *Semantic (logic) errors* – syntactically correct, but mistakes in specifics and/or order

Up Next

- Ch. 3 – Functions