Writing Simple Programs

Theresa Migler-VonDollen
CMPS 5P
The process of creating a program is often broken down into stages according to the information that is produced in each phase.

- **Analyze the Problem** Figure out exactly the problem to be solved. Try to understand it as much as possible.
- **Determine Specifications** Describe exactly what your program will do.
  - Don’t worry about how the program will work, but what it will do.
  - Includes describing the inputs, outputs, and how they relate to one another.
Analyze the Problem

Determine Specifications

Create a Design
  ▶ Formulate the overall structure of the program.
  ▶ This is where the how of the program gets worked out.
  ▶ You choose or develop your own algorithm that meets the specifications.

Implement the Design
  ▶ Translate the design into a computer language (Python).
The Software Development Process

- Analyze the Problem
- Determine Specifications
- Create a Design
- Implement the Design
- Test/Debug the Program
  - Try out your program to see if it worked.
  - If there are any errors (bugs), they need to be located and fixed. This process is called *debugging*.
  - Your goal is to find errors, so try everything that might "break" your program!
The Software Development Process

- Analyze the Problem
- Determine Specifications
- Create a Design
- Implement the Design
- Test/Debug the Program
- Maintain the Program
  - Continue developing the program in response to the needs of your users.
  - In the real world, most programs are never completely finished - they evolve over time.
The Software Development Process

- Analyze the Problem
- Determine Specifications
- Create a Design
- Implement the Design
- Test/Debug the Program
- Maintain the Program

Let’s try it!
Write a Python program that asks the user for a temperature in Celsius and tells the user what the temperature is in Fahrenheit.
Temperature Converter

- **Analyze the Problem**
  - The temperature is given in Celsius, user wants it expressed in degrees Fahrenheit.

- **Determine Specifications**
  - Input - temperature in Celsius
  - Output - temperature in Fahrenheit
  - Output = \( \frac{9}{5} \text{(Input)} + 32 \)
Create a Design

- Input, Process, Output (IPO)
- Prompt the user for input (Celsius temperature)
- Process it to convert it to Fahrenheit using
  \[ F = \frac{9}{5}(C) + 32 \]
- Output the result by displaying it on the screen.

Before we start coding, let’s write a rough draft of the program in pseudocode.

Pseudocode is precise English that describes what a program does, step by step.

Using pseudocode, we can concentrate on the algorithm rather than the programming language.
Create a Design

- Pseudocode:
  - Input the temperature in degrees Celsius (call it celsius)
  - Calculate fahrenheit as \((9/5) \times celsius + 32\)
  - Output fahrenheit
Temperature Converter

- Implement the Design
- Test the Program
  - Does the program work for all numbers? Decimals? Negative numbers?
- Maintain the Program
  - Do you anticipate any additions a user of this program might have?
  - Perhaps an extra check to see if the temperature is below freezing?
Elements of the Program

- **Names**
  - Names are given to variables (celsius, fahrenheit), modules (main, convert), etc.
  - These names are called identifiers
  - Every identifier must begin with a letter or underscore, followed by any sequence of letters, digits, or underscores.
  - Identifiers are case sensitive.
  - Some identifiers are part of Python itself. These identifiers are known as reserved words. This means they are not available for you to use as a name for a variable, etc. in your program.
  - and, del, for, is, raise, assert, elif, in, print, etc.
  - For a complete list, see table 2.1
Expressions

- The fragments of code that produce or calculate new data values are called expressions.
- Literals are used to represent a specific value, e.g. 3.9, 1, 1.0
- Simple identifiers can also be expressions.
- Simpler expressions can be combined using operators.
- +, −, *, /, **
- Spaces are irrelevant within an expression.
- The normal mathematical precedence applies.
Output Statements

- A `print` statement can print any number of expressions.
- Successive `print` statements will display on separate lines.
- A bare `print` will print a blank line.
Assignment Statements

Simple Assignment:

\(<variable> = <expr>\)

variable is an identifier, expr is an expression

The expression on the right-hand side is evaluated to produce a value which is then associated with the variable named on the left-hand side.

For example: \(fahrenheit = \frac{9}{5} \times celsius + 32\)

Variables can be reassigned.

Variables are like a box we can put values in.

When a variable changes, the old value is erased and a new one is written in.

NOT exactly
Elements of the Program

- Assignment Statements
  - Python doesn’t overwrite these memory locations (boxes).
  - Assigning a variable is more like putting a sticky note on a value and saying, “this is x”.

```
Before                     After
 x  =  x  +  1
10                             11
```

Assignment Input

- The purpose of an input statement is to get input from the user and store it into a variable.
- `<variable> = eval(input(<prompt>))`
- First the prompt is printed
- The input part waits for the user to enter a value and press <enter>
- The expression that was entered is evaluated to turn it from a string of characters into a Python value (a number).
- The value is assigned to the variable.
Simultaneous Assignment

- Several values can be calculated at the same time.
- `< var >, < var >, ... = < expr >, < expr >, ...`
- Evaluate the expressions on the right-hand side and assign them to the variables on the left-hand side.
- `sum, diff = x + y, x - y`
- How could you use this to swap the values for `x` and `y`?

  - Why doesn’t this work?
    
    ```
    x = y
    y = x
    ```

  - We could use a temporary variable...
  - But it’s even easier in Python:
    ```
    x, y = y, x
    ```
Elements of the Program

- **Definite Loops**
  - A *definite loop* executes a definite number of times, i.e., at the time Python starts the loop it knows exactly how many iterations to do.
    - `for <var> in <sequence>:`
      - `<body>`
  - The beginning and end of the body are indicated by indentation.
  - The variable after the for is called the *loop index*. It takes on each successive value in sequence.
  - `range` is a built-in Python function that generates a sequence of numbers, starting with 0.
  - `list` is a built-in Python function that turns the sequence into an explicit list.
Write a Python program that takes an amount of money (the principal) and an apr and calculates the amount of the investment after 10 years.
Practice with Future Value Program

- **Analysis**
  - Money deposited in a bank account earns interest.
  - How much will the account be worth 10 years from now?
  - Inputs: principal, interest rate
  - Output: value of the investment in 10 years
Practice with Future Value Program

- **Specification**
  - User enters the initial amount to invest, the principal.
  - User enters an annual percentage rate, the interest.
  - The specifications can be represented like this:
    - **Program**: Future Value
      - **Inputs**: principal: the amount of money being invested, in dollars
        apr: the annual percentage rate expressed as a decimal number.
    - **Output**: The value of the investment 10 years in the future
      - **Relationship**: Value after one year is given by \( \text{principal} \times (1 + \text{apr}) \). This needs to be done 10 times.
Practice with Future Value Program

» Design

» Print an introduction
  Input the amount of the principal (principal)
  Input the annual percentage rate (apr)
  Repeat 10 times:
  \[
  \text{principal} = \text{principal} \times (1 + \text{apr})
  \]
  Output the value of principal

23 / 25
Implementation

- Each line translates to one line of Python (in this case)
- Print an introduction
  ```python
  print("This program calculates the future value")
  print("value of a 10-year investment.")
  ```
- Input the amount of the principal:
  ```python
  principal = eval(input("Enter the initial principal: "))
  ```
- Input the annual percentage rate
  ```python
  apr = eval(input("Enter the annual interest rate: "))
  ```
- Repeat 10 times: for i in range(10):
  ```python
  Calculate principal = principal * (1 + apr)
  ```
- Output the value of the principal at the end of 10 years
  ```python
  print("The value in 10 years is:", principal)
  ```
Practice with Future Value Program

▶ Debug
▶ Maintain
  ▶ Are there any immediate extensions or improvements that we can make to our program?
    ▶ Calculate the value of the investment for a variable amount of years.
    ▶ Allow for semianual compounding.