CITS 4406
Problem Solving & Programming

Tim French

Lecture 02 – The Software Development Process

(These slides are based on John Zelle’s powerpoint slides for lectures accompanied with the text book)
Objectives of this lecture

- To know the steps of a software development process
- Understand and write simple Python statements
- Output information to the screen
- Assign values to variables
- Input numeric information from the user through keyboard
The Software Development Process

1. Analyze the Problem

2. Determine the Specifications

3. Create a Design

4. Implement the Design i.e. write the program

5. Test/Debug the Program

6. Maintain the Program
Analyze the Problem

- Figure out what exactly is the problem to be solved.
- Try to understand it as much as possible.
- You cannot solve a problem unless you fully understand it.
Determine Specifications

- Describe exactly what your program will do
  - At this stage, don’t worry how it will do it.
  - Only figure out what your program will do.

- Describe the inputs and outputs.

- Describe how the outputs relate to the inputs.
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 solves the problem and meets the specifications.
Implement the Design

- Translate the design into a computer language.

- Write each step of the design as program statements.

- In CITS1401, we will use Python as our programming language.
Test/Debug the Program

- Try out your program to see if it works as expected.

- There could be syntax errors or errors in how you designed your algorithm.

- 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!

- Try different input values and see if the results are correct.
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.
Example Program: Temperature Converter

Analysis – the temperature is given in Celsius, user wants it expressed in degrees Fahrenheit.

Specification

• Input – temperature in Celsius
• Output – temperature in Fahrenheit
• Output = \( \frac{9}{5}(input) + 32 \)
Example Program: Temperature Converter

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
Write the Pseudocode First

- Before writing the actual program (code), let’s start by writing the 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.
Pseudocode

Pseudocode:

1. Prompt the user to input the temperature in degrees Celsius (call it celsius)
2. Calculate fahrenheit as \((9/5)\times\text{celsius}+32\)
3. Output fahrenheit

Now we need to convert this to Python!
Example Program: Temperature Converter

```python
#convert.py
# A program to convert Celsius temps to Fahrenheit
# by: Susan Computewell

def main():
    celsius = eval(input("What is the Celsius temperature? "))
    fahrenheit = (9/5) * celsius + 32
    print("The temperature is ",fahrenheit," degrees Fahrenheit.")

main()
```
Testing the program

The next step is to test the program

```python
>>> What is the Celsius temperature? 0
The temperature is 32.0 degrees Fahrenheit.
>>> main()
>>> What is the Celsius temperature? 100
The temperature is 212.0 degrees Fahrenheit.
>>> main()
>>> What is the Celsius temperature? -40
The temperature is -40.0 degrees Fahrenheit.
>>> 
```
Elements of 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.
Elements of Program

These are all different, valid names

- X
- Celsius
- Spam
- spam
- spAm
- Spam_and_Eggs
- Spam_And_Eggs
Elements of Program

- Some identifiers are part of Python itself.

- These identifiers are known as reserved words. 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
Elements of Program

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.
Elements of Program

```python
>>> x = 5
>>> x
5
>>> print(x)
5
>>> print(spam)
Traceback (most recent call last):
  File "<pyshell#15>", line 1, in -toplevel-
    print spam
NameError: name 'spam' is not defined
```

NameError is the error when you try to use a variable without a value assigned to it.
Elements of Program

- Simpler expressions can be combined using operators.

- `+, -, *, /, **`

- Spaces are irrelevant within an expression.

- The normal mathematical precedence applies.

- `((x1 - x2) / 2*n) + (spam / k**3)`
Elements of Program

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.
Elements of Program

```python
print(3+4)  # 7
print(3, 4, 3+4)  # 3 4 7
print()  #
print(3, 4, end=" ")  # 3 4
print(3 + 4)  # 7
print("The answer is", 3+4)  # The answer is 7
```
Assignment Statements

- Simple Assignment

- `<variable> = <expr>`
  variable is an identifier, expr is an expression

- The expression on the RHS is evaluated to produce a value which is then associated with the variable named on the LHS.
Assignment Statements

\[ x = 3.9 \times x \times (1-x) \]

\[ \text{fahrenheit} = \frac{9}{5} \times \text{celsius} + 32 \]

\[ x = 5 \]
Assignment Statements

Variables can be reassigned as many times as you want!

```python
>>> myVar = 0
>>> myVar
0
>>> myVar = 7
>>> myVar
7
>>> myVar = myVar + 1
>>> myVar
8
>>> 
```
Assignment Statements

- 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.

\[
x = x + 1
\]

Before \( x = 10 \)

After \( x = 11 \)
Assignment Statements

- Technically, this model of assignment is simplistic for Python.

- 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”.
Assigning Input

- The purpose of an input statement is to get input from the user and store it into a variable.

- `<variable> = eval(input(<prompt>))`
Assigning Input

- 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 in the RHS and assign them to the variables on the LHS
Simultaneous Assignment

- 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…
Simultaneous Assignment

We can swap the values of two variables quite easily in Python!

```python
x, y = y, x
>>> x = 3
>>> y = 4
>>> print x, y
3 4
>>> x, y = y, x
>>> print x, y
4 3
```
Simultaneous Assignment

- We can use the same idea to input multiple variables from a single input statement
- Use commas to separate the inputs

```
Def spamnegges():
    spam, eggs = eval("Enter # of spam slices followed by # of eggs:")
    print("You ordered", eggs, "eggs and ", spam, "slices of spam.")
```

```python
>>> spamnegges()
Enter # of spam slices followed by # of eggs: 3, 2
You ordered 2 eggs and 3 slices of spam.
```
Definite Loops

- A definite loop executes a definite number of times

- When Python starts, it knows exactly how many iterations to do (how many times to execute the statements inside the loop)

- for <var> in <sequence>:
  <body>

- The beginning and end of the body are indicated by indentation.
Definite Loops

- `for <var> in <sequence>:`
  - `<body>

- The variable after the “for” is called the “loop” index. It takes on each successive value in sequence.
Definite Loops

```python
>>> for i in [0, 1, 2, 3]:
    print (i)
0
1
2
3

>>> for odd in [1, 3, 5, 7]:
    print(odd*odd)
1
9
25
49
```
Definite Loops

- In `chaos.py`, what did `range(10)` do?

  ```python
  >>> list(range(10))
  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
  ```

- `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.

- The body of the loop executes 10 times.
Definite Loops

_for_ loops alter the flow of program execution, so they are referred to as _control structures._
Example Program: Future Value

Analysis

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

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 …
Example Program: Future Value

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 $principal \times (1 + apr)$. This needs to be done 10 times.
Example Program: Future Value

- **Design**
  - Print an introduction
  - Input the amount of the principal (principal)
  - Input the annual percentage rate (apr)
  - Repeat 10 times:
    principal = principal * (1 + apr)
  - Output the value of principal
Example Program: Future Value

Implementation

- Each line translates to one line of Python (in this case)

- Print an introduction
  ```python
  print ("This program calculates the future")
  print ("value of a 10-year investment.")
  ```

- Input the amount of the principal
  ```python
  principal = eval(input("Enter the initial principal: "))
  ```
Example Program: Future Value

• Input the annual percentage rate
  apr = eval(input("Enter the annual interest rate: "))

• Repeat 10 times:
  for i in range(10):
    principal = principal * (1 + apr)

• Calculate principal = principal * (1 + apr)

• Output the value of the principal at the end of 10 years
  print("The value in 10 years is:", principal)
Example Program: Future Value

# futval.py
# A program to compute the value of an investment carried 10 years into the future

def main():
    print("This program calculates the future value of a 10-year investment.")

    principal = eval(input("Enter the initial principal: "))
    apr = eval(input("Enter the annual interest rate: "))

    for i in range(10):
        principal = principal * (1 + apr)

    print("The value in 10 years is:", principal)

main()
Example Program: Future Value

```python
>>> main()
This program calculates the future value of a 10-year investment.
Enter the initial principal: 100
Enter the annual interest rate: .03
The value in 10 years is: 134.391637934

>>> main()
This program calculates the future value of a 10-year investment.
Enter the initial principal: 100
Enter the annual interest rate: .10
The value in 10 years is: 259.37424601
```
Lecture Summary

- Understanding the concept of assignment in Python

- We learned how to
  - assign values to variables
  - do multiple assignments in one statement
  - definite simple definite loops

- “for” loop alters the sequence of the program
Lecture Summary

- We learned about the steps of a software development process
- We wrote and analysed simple Python statements
- We learned the concept of pseudocode
- We learned about the elements of a program