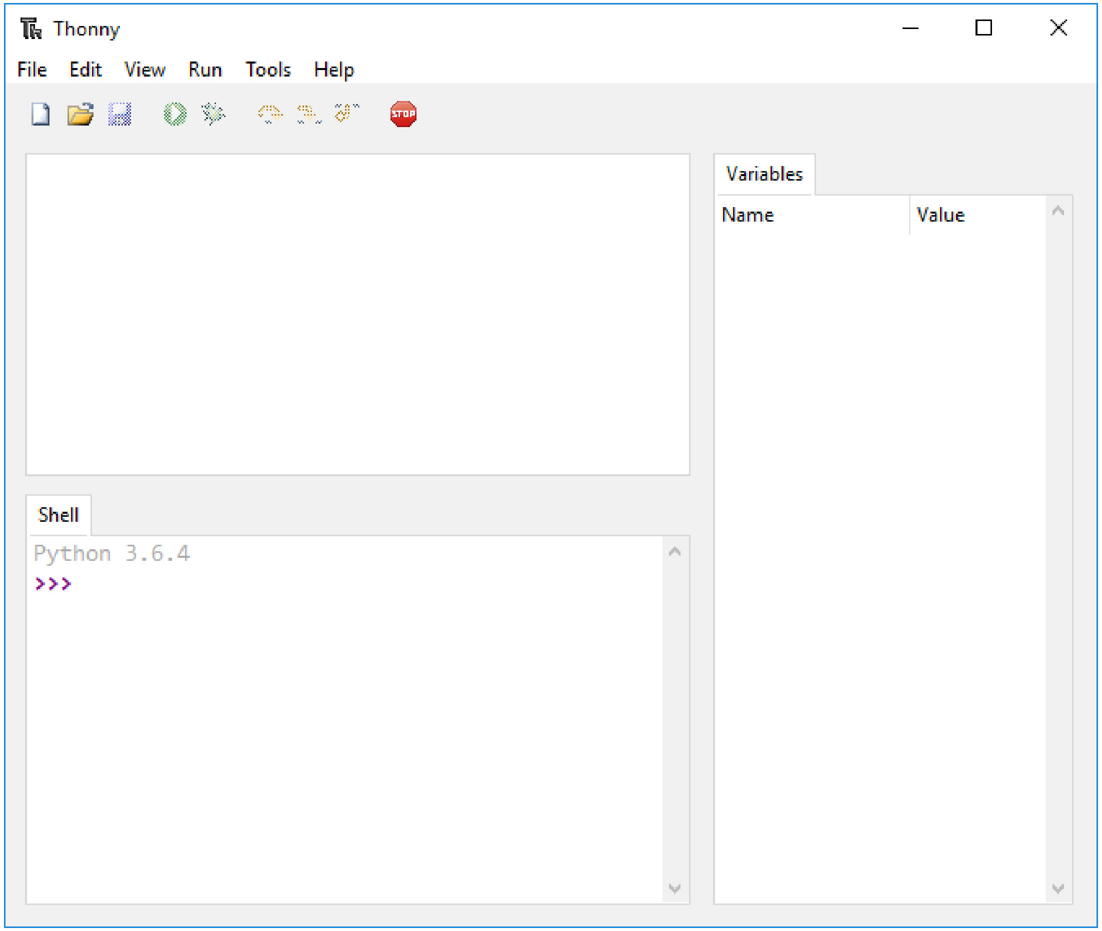
THE UNIVERSITY OF WESTERN AUSTRALIA

Lecture 2
The Software Development Process

Objectives of This Lecture

- Getting started with Python/Thonny
- To know the steps of a software development process
- Understand and write simple Python statements
- Understand the concept of pseudocode
- Elements of a program

Getting stated with Thonny



```
Python 3.6.4
>>>
```

Getting Started with Python

Start with single statements

```
>>> 2+3
5
>>> 22/7
3.142857142857143
>>> 3**2
9
>>> print(“Hello world”)
Hello world
>>> print(“2+3=“, 2+3)
2+3=5
```

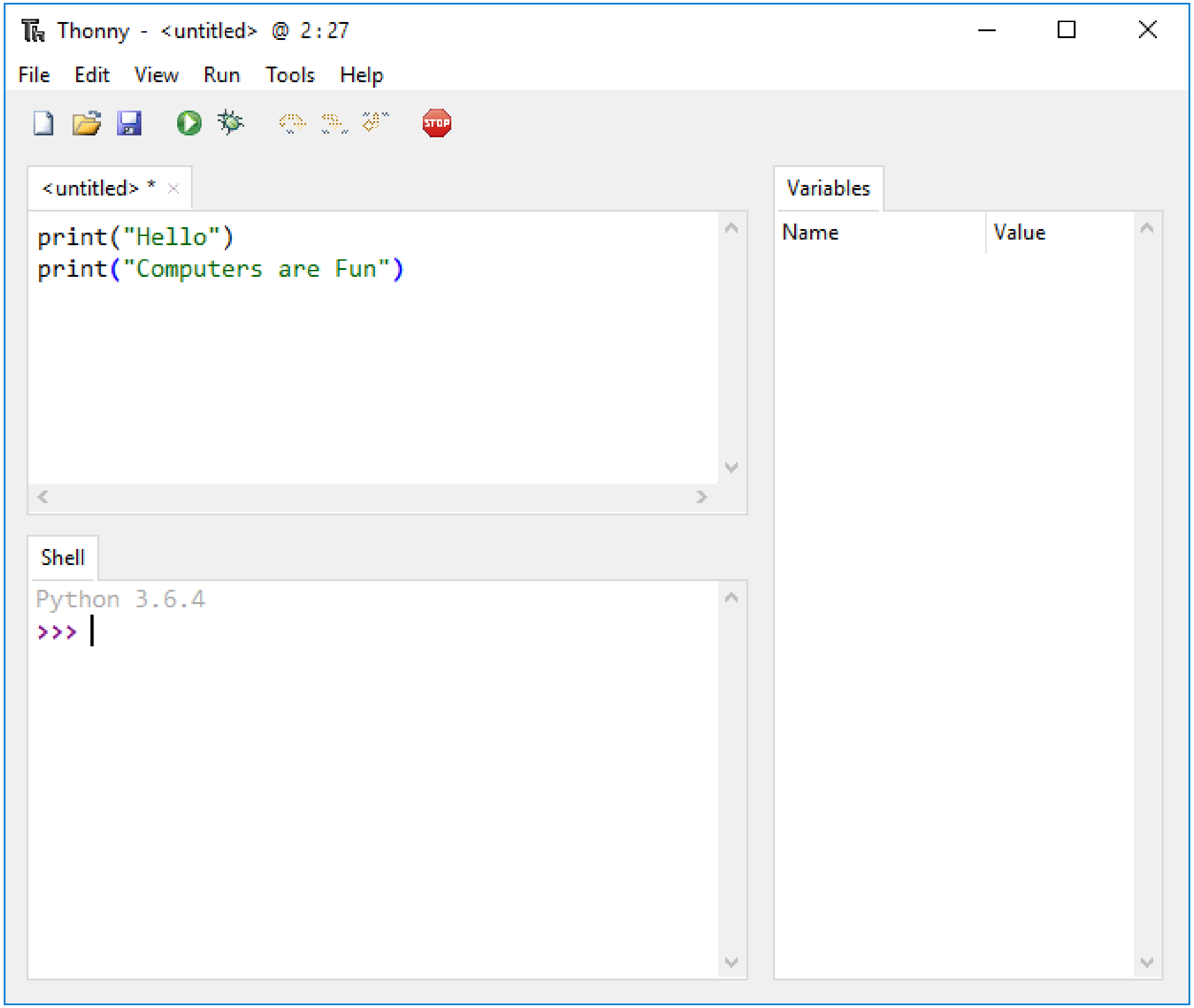
Group Multiple Statements

- To solve a problem, we generally need to execute more than one statements.
  - *One way to do this is to use a file*
  - *Create a file and type the statements*

```
print("Hello")
print("Computers are Fun")
```

- Press the green button to run the file
- All statements will be executed line by line

Thonny demo



```
print("Hello")
print("Computers are Fun")
```

```
Python 3.6.4
>>> 
```

Analyse 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.

  ⇒ Talk to users. Better still, *listen* to users

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.
  - *Not code yet.*
- 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.
  - *You will be working individually in this unit, but in industry, teams are involved in projects*
- We will use Python 3 as our programming language.

Test/Debug the Program (Important)

- Your program will often have syntax errors.
  - *These are highlighted by the interpreter.*
  - *Need to fix them before the program will work at all.*

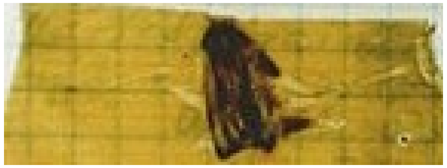
```
>>> 32/
  File "<pyshell>", line 1
    32/
      ^
SyntaxError: invalid syntax
```

- Even syntactically correct programs may not work as expected.
  - *Logic errors – the sequence of instructions are legal, and the program will run, but do not compute the intended function (semantic error)*
  - *For multiplication of two number 2 & 5*

```
>>> 2**5
32
```

Debugging

- If there are any errors (*bugs*), they need to be located and fixed. This process is called debugging.



- **THOROUGH TESTING IS CRUCIAL.**
  - *If you don't find the bugs, the users will !!!*
  - *Your goal is to find errors, so try everything that might "break" your program!*

    ⇒ Antibugging (putting in tests for likely errors)
  - *Try different input values and see if the results are correct.*
  - *Important in industry. More immediately, important for the Projects*

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.
  - *Software Life Cycle*

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 = 9/5(input) + 32*

Temperature Converter: Design

Design

- *Overall: Input, Process, Output (IPO)*
- *Prompt the user for input (Celsius temperature)*
- *Process it to convert it to Fahrenheit using* $F = 9/5(C) + 32$
- *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.
- Difference between algorithm and pseudocode
  - *Algorithms can be described in various ways, from pure mathematical formulas to complex graphs, more times than not, without pseudocode.*
  - *Pseudocode describes how you would implement an algorithm without getting into syntactical details*

Pseudocode

Pseudocode

```
1. Prompt the user to input the temperature in degrees
   Celsius (store it as celsius)
2. Calculate fahrenheit as $(9/5)*celsius+32$
3. Output fahrenheit
```

Now we need to convert this to Python!

Temperature Converter: Python program

```
""" convert.py
 A program to convert Celsius temps to Fahrenheit
 by: Someone Programmer """

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

- Note the multiline comment at the start. It is important as it tells the maintainer:
  - What the program does
  - Statement of authorship

Testing the Program

The next step is to test the program (Press Run or green button on Thonny)

```
>>>
What is the Celsius temperature? 0
The temperature is  32.0  degrees Fahrenheit.
>>>
What is the Celsius temperature? 100
The temperature is  212.0  degrees Fahrenheit.
>>>
What is the Celsius temperature? -40
The temperature is  -40.0  degrees Fahrenheit.
>>>
```

Elements of Program: Identifiers

- Names
  - *Names are given to:*
    - variables (e.g. celsius, fahrenheit)
    - functions (e.g. main)
    - modules (e.g. temp_converter, chaos)

    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.*

Identifiers examples

- These are all different, valid names
  - *X*
  - *Spam*
  - *spam*
  - *spAm*
  - *Spam_and_Eggs*
  - *Spam_And_Eggs*
  - *_x*
  - *C3P0*

Reserved words

- 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, def, for, is, raise, assert, elif, in, print,` *etc*.
- For a complete list, see the link for more! https://www.w3schools.com/python/python_ref_keywords.asp

Elements of Program: Expressions

- The fragments of code that produce or calculate new data values are called *expressions*.

```
(9/5) * celsius + 32
```

- Expressions are composed of literals, variables and operators

- *Literals* are used to represent a specific value, e.g. `3.9`, `-1`, `1.0`, `3.0e8`, `“Fred”`

- Two expressions can be combined with an operator to make another expression

Elements of Program: Statement

- A standalone unit of execution that can be of one or several lines of code is called *statement*

```
fahrenheit =(9/5) * celsius + 32

print("The temperature is ",fahrenheit," degrees Fahrenheit.")
```

- Statements can include expressions

Elements of Program

```
>>> x = 5
>>> x          # This only works on interactive interpreter
5
>>> print(x)   # This works both interactive and from file
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 first having a value having been assigned to it.

Mathematical operators

- Simpler expressions can be combined using *operators*.
- `+,   -,   *,   /, //, **`
- Spaces are irrelevant within an expression
  - *But readability!!*
- The normal mathematical precedence applies.
- `((x1 - x2) / 2*n) + (spam / k**3)` same as `(x1 - x2) / 2*n + spam / k**3`

Elements of Program: Input Information

- The `input` function prints text and expects a value (actually a string typed by the user)

```
z = input('type a value ')
```

- The `int` function converts a string of digits to an integer; it will throw an exception (error) if the user did not type an integer

```
z = int(input('type a value '))
```

- The `float` function works the same way, but expects a floating (decimal) point number

Elements of Program: Output

- Output Statements
  - *A* `print` *function can print any number of expressions (separated by commas).*
  - *Successive print statements will display on separate lines.*
  - *A bare print will print a blank line.*

print() function

Expression	Produces
`print(3+4)`	`7`
`print(3, 4, 3+4)`	`3 4 7`
`print()`	
`print(3 + 4)`	`7`
`print("The answer is", 3+4)`	`The answer is 7`

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 importance of testing
- We learned about the elements of a program