KIT100 PROGRAMMING PREPARATION

Lecture Two:

Why Program?

Introductory Programming Concepts

Lecture Objectives

- Computing Tools and Terms
- Why program?
- What is a programming language?
- Programming Tools and Terms
- What is Python?
- Using Python

Computing Tools & Terms

- Components of a computer
- How those components interact
- How computers store and manipulate information
- Operating System

Hardware & Software

- Hardware
  - Physical, tangible parts of a computer
    - keyboard, monitor, wires, chips
- Software
  - programs and data
    - a program is a series of instructions
    - data is what the program instructs the computer to manipulate (numbers, words etc)
- A computer requires both hardware and software

Hardware – main architecture

Hardware
- Computer Basic Architecture

Chip that executes program commands

Central Processing Unit (CPU)

For example, an intel i7 processer @ 3.33GHz (3.33 billion clock cycles a second) could execute **177,730 million instructions per second (MIPS)** at peak performance.

CPU interacts with main memory

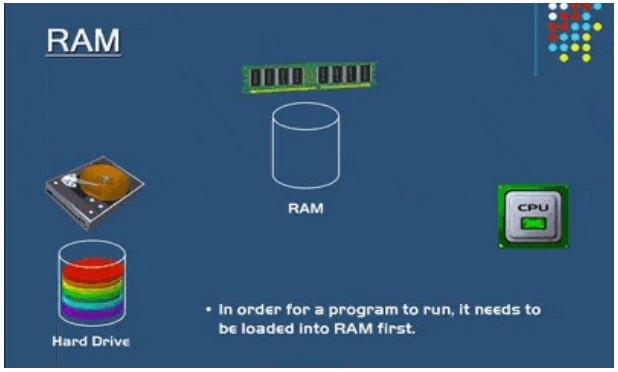
Primary storage area for programs and data that are in active use

Main Memory

Hardware – Secondary Memory Devices

Central Processing Unit

Information is moved between main memory and secondary memory as needed



Main Memory

Hard Disk

USB Disk

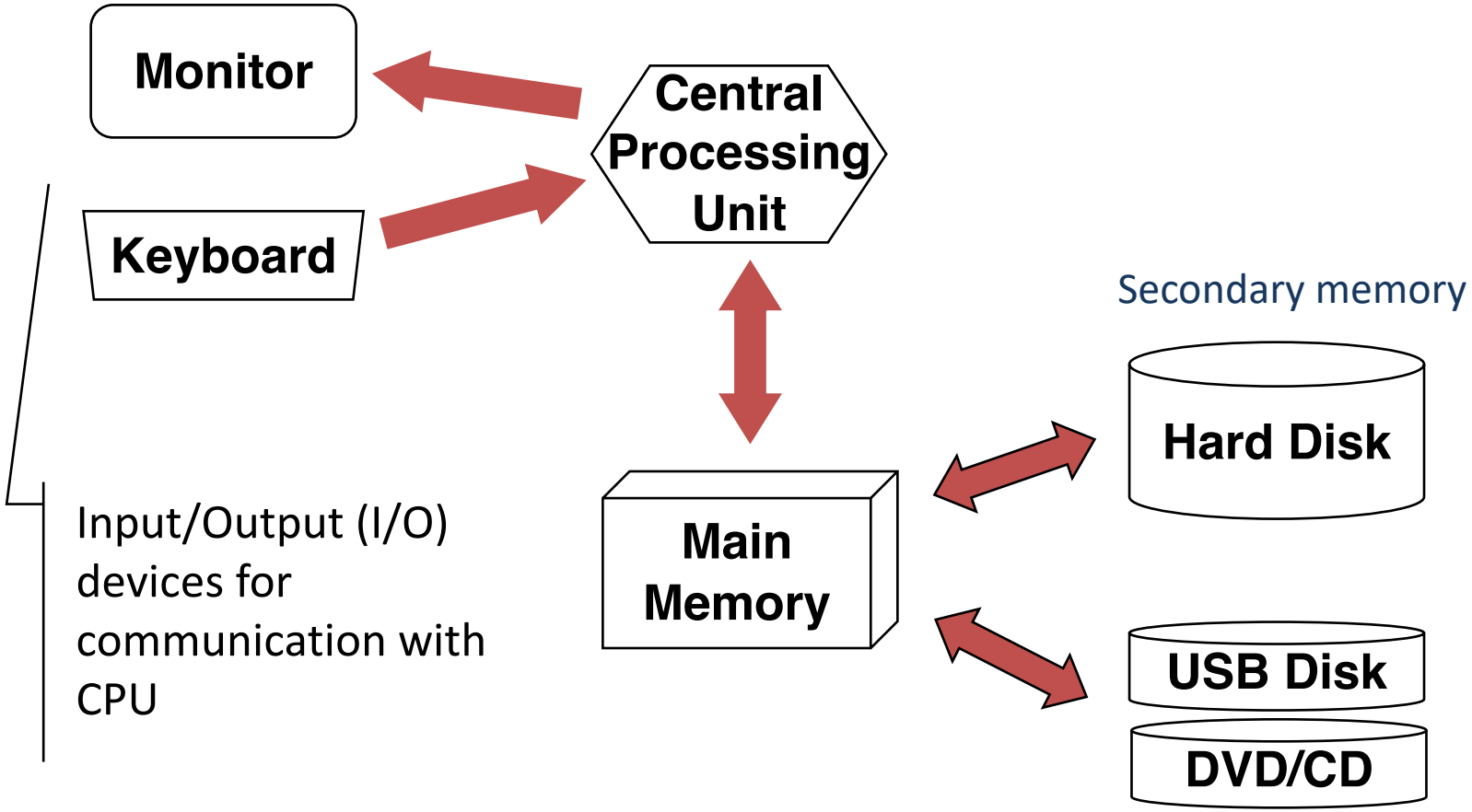
DVD/CD

- Called “Secondary memory” or “External memory”
- Long-term storage
- Interact slow

- Also called “Internal memory” or “random-access memory (RAM)”
- Short-term storage
- Interact very fact

Typical Hardware Configuration

Computer Main Architecture

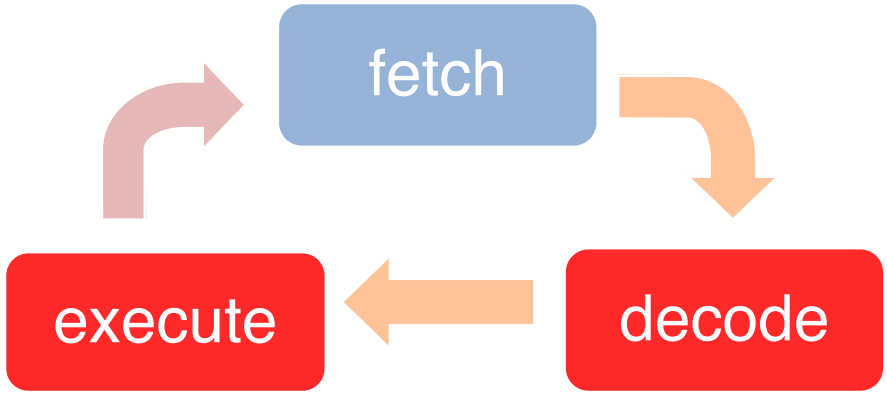


The Central Processing Unit (CPU)

A **Central Processing Unit (CPU)** is also called a microprocessor or core

It/they continuously follow(s) the **fetch-decode-execute cycle**:

Retrieve an instruction from “main memory”



Determine what the instruction is

Carry out the instruction
- access and modify data if instructed to

Software Categories

Software Categories:

- Operating System (OS) (e.g., Win/Mac/Linux)
  - controls all machine activities
  - user interface to the computer
  - manages resources such as the CPU and memory
  - Files
  - Programs – e.g. editor, compiler
- Application program (App) (e.g., Office365)
  - generic term for any other kind of software
  - we will write applications
- Graphical user interface (GUI) (e.g., desktop)
  - some of our programs will have a GUI

Applications

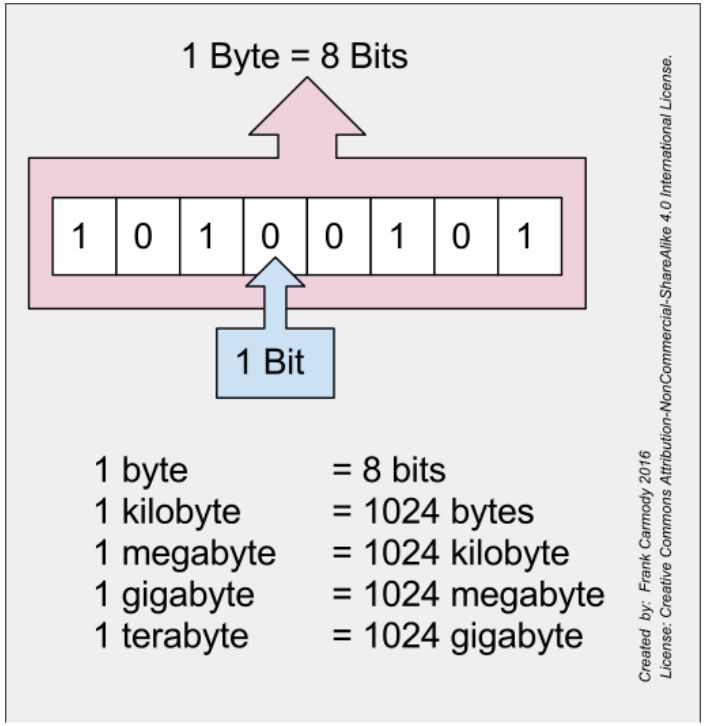
- Ultimately, any application we create will interact (successfully or unsuccessfully) with the computer and the data it contains.
- Any application manipulates data in some way.
  - **C**reate
  - **R**ead
  - **U**pdate
  - **D**elete

How computers store data ?

- All data that is stored within a computer is converted to 0's and 1's
- The data is stored in the computer's memory
- The memory is divided into storage locations known as **bytes**

- Letter, numbers, and symbols are stored as binary numbers (combinations of 0's and 1's with base of 2, instead of 10)

  - $00000001_2 = 1_{10}$
  - $00000010_2 = 2_{10}$
  - $01011011_2 = 91_{10}$

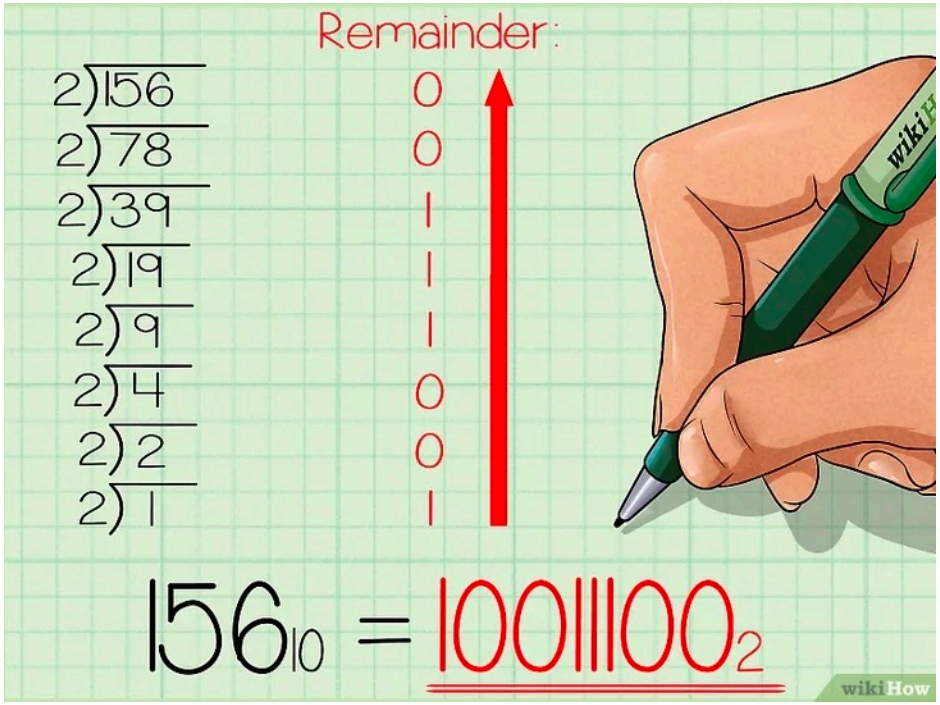


Example

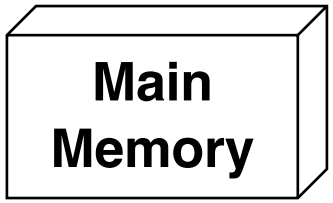
Converting **from decimal (10) to binary (2),**

We are now using **2** as the divisor,

- If the result is **even**, the remainder is recorded as **0**;
- if the result is **odd**, the remainder is recorded as **1**.



Memory

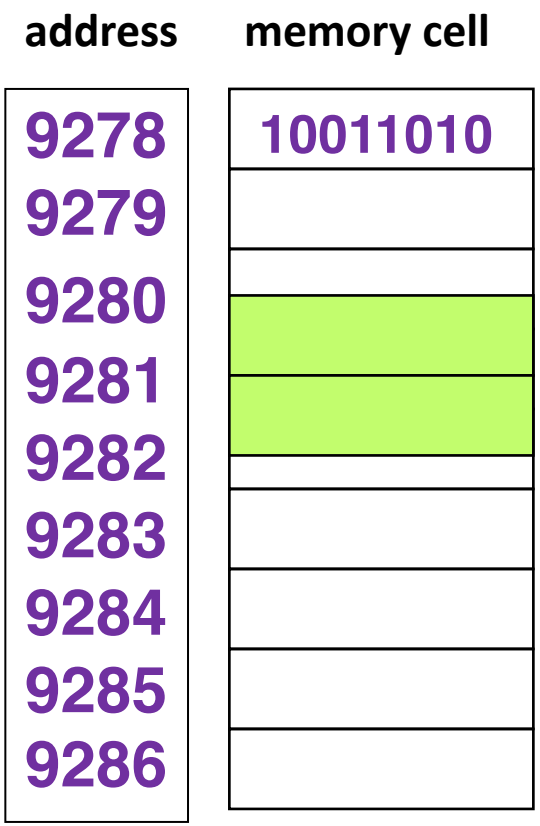


Main memory is divided into many memory locations (or cells)

address	memory cell
9278	
9279	
9280	
9281	
9282	
9283	
9284	
9285	
9286	

Each **memory cell** has a numeric **address**, which uniquely identifies it, often these are numbers in hexadecimal (16 bits), e.g. 9a3f

Storing Information



Each memory cell stores a set number of bits (usually 8 **bits**, or one **byte**) called a word

More complex data needs to use more than one memory location.

Large values are stored in consecutive memory locations

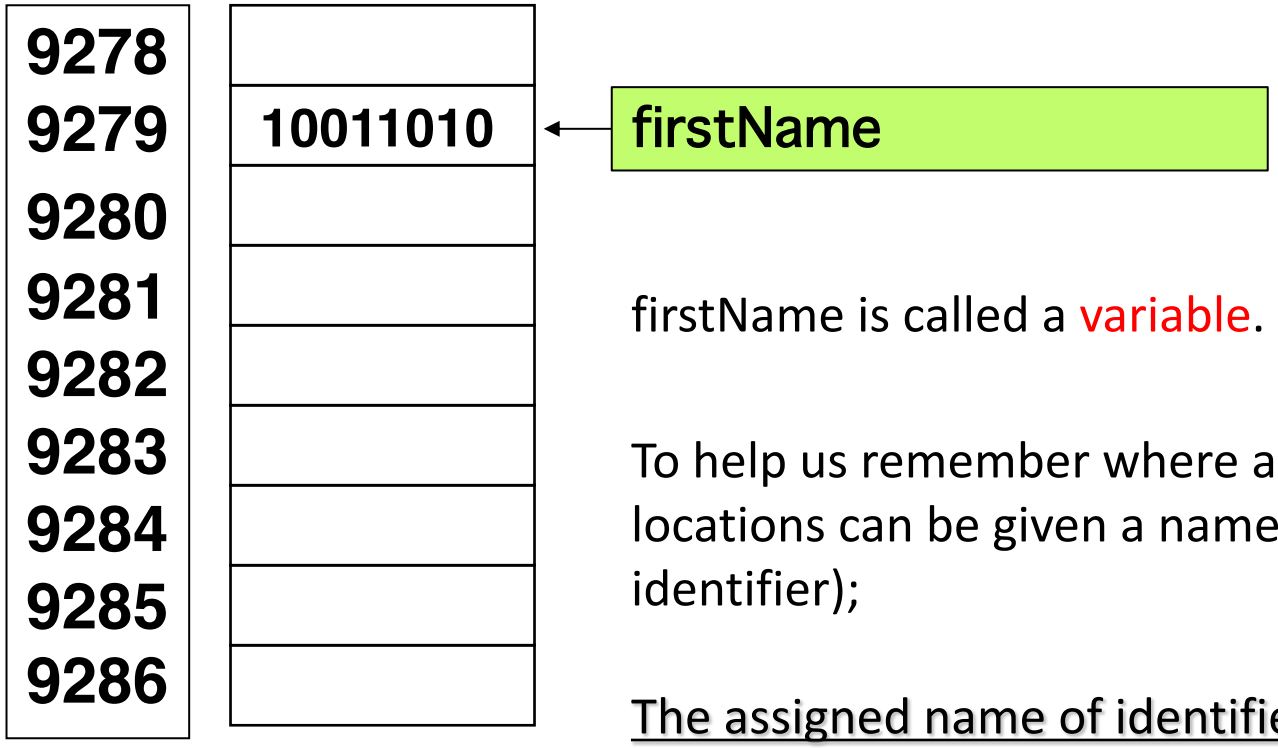
How can we then represent the number 256?

- maximum number we can store is 255 (11111111_2)
- Use two bytes to represent integers!

Storing Information

Humans don't deal well with having to remember many arbitrary numbers

- We assign **labels** to memory locations.
- These labels are called ***identifiers***.



firstName is called a variable.

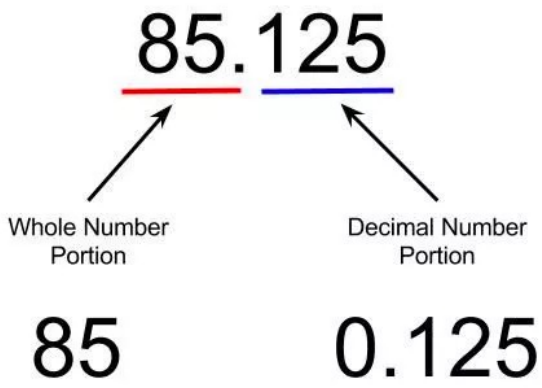
To help us remember where a value is, memory locations can be given a name (which is called an identifier);

The assigned name of identifier is 'variable'.

Storing Information

So variables contain different **types** of data,

- **Texts** (strings) - such as 'yourname'.
- **Whole numbers** (integers) – such as 85
- **Decimal numbers** (floats) – such as 0.125
- **True/false values** (Boolean) – such as 10011010



9278	
9279	10011010
9280	
9281	
9282	
9283	
9284	
9285	
9286	

firstName (String)

firstName is a variable of type "**String**"

To help us understand the value, variables are associated with a type

Why Program?

- Things that humans might find hard, or boring, computers are actually good at
- Relevant to most fields of expertise:

  - Recording Data
  - Analysing Data
  - Finding Patterns
  - Plotting Data
  - Automating Tasks
  - Comparing Data
  - Multi-Tasking
  - Image Processing

Why Program?

- As such computer programs are used not only in our leisure time with games and social networks, but also in e.g.
  - Biology, Chemistry, Physics, Bioinformatics
  - Commerce, Health, Industry
  - Sociology, Law, Media Production
- Basically computers are used in every field, and often programs are written by people whose main field of expertise is not computer programming
- To create successful applications that contain procedures so the computer can communicate with us and know what we want it to do!

What is a Programming Language?

- A programming language specifies
  - words and symbols that can be used to write a program
  - **rules** to form valid program statements
- Source code (text) written by programmer
  - Can just be typed into the editor and run; or can be stored in a file and loaded into memory to run
  - A program usually consists of one or more *classes*; Classes consist of one or more *functions*; Functions consist of one or more *statements*

Types of Programming Languages

- Machine Language
  (purely binary)
- Assembly Language
  (Low Level Language, symbols)
- High-level Programming Language
  (near-English terms)

CPU-readable

Human-readable

- ✓ CPUs ONLY understand machine language.
- ✓ Some programs used to be developed in **assembly language**.
- ✓ Assembler is using an intermediate (automatic) step between converting high-level language to CPU-specific machine code.

Machine Languages

- Machine Language
  - A computer's native language – a set of primitive instructions.
  - In the form of **binary code**

    E.g.

    `1101101010011010` - representing *"clear register A"*

    `1010110100110111` - representing *"add one to register A"*
  - Specific to the actual CPU the program is running on

    E.g. Intel x86 architecture, ARM architecture

Assembly Languages

- Programming in Machine Language is boring and almost impossible for humans to understand and follow!
- Assembly Language was created to make life a bit easier.
- Uses short descriptive words, ***mnemonics***, to represent each of the machine language instructions.

  E.g.

  `LDA 2` – representing "*Load register A with 2*"

  `STAB` – representing "*Store register A in memory address stored in register B*"

  – Specific to the actual CPU the program is running on

High-Level Languages

- English-like (most common, but programming languages based on other human languages exist e.g. programming in Chinese)
- Easier to learn and use than assembly language
- Instructions in high-level language are called statements
- High level languages are written in source code (text) and need to be **translated** using other programming tools called interpreters or compilers, to machine code to run on a CPU.
  - An **interpreter** reads one statement at a time from the source code, translates it to the machine code and **then executes it** (ask the compute to do sth. one by one)
  - A **compiler** translates the **entire source code** into machine code that can be executed later. (ask the compute to do all sth.)

High-Level Languages

- C
- Visual Basic
- Java
- C++
- C#
- Python

Programming Tools & Terms

- Syntax
- Semantics
- Structure of a program
- Program development cycle

Syntax and Semantics

- A program will only run if it **exactly** follows the rules of the language (**syntax**) - similar as the grammar human use.
  - The syntax rules of a language define how symbols, reserved words, and identifiers may be combined together in source code to make a valid program

- A program will always do what we tell it to do, not what we meant to tell it to do (**semantics**)
  - The semantics of a program statement define what that statement means (what it will do when it is executed)

- A program that is **syntactically** correct is not necessarily logically (semantically) correct.

- *You can create a program that correctly follows the syntax rules, but doesn't do what you thought it would do – usually incorrect assumptions or human error is to blame*

Structure of a program

Think of writing a letter or email:

- A written language specifies
  - words and punctuation that can be used to write a letter
  - rules to form understandable and recognisable statements
  - ‘whitespace’ doesn’t change the meaning of the letter
- Paper/Email — contains the letter
  - Consists of sections (subject, body, signature)
  - Body consists of paragraphs; Paragraphs consist of sentences

A standard letter construction

Re: My recent dummy-spit

Dear Mummy,

I am writing because I cannot find my teddy. It was sitting next to my beer, and I can no longer find it.

If you have stolen it, could you please return it? The other guys in the regiment all have theirs… Thanks.

Sincerely,

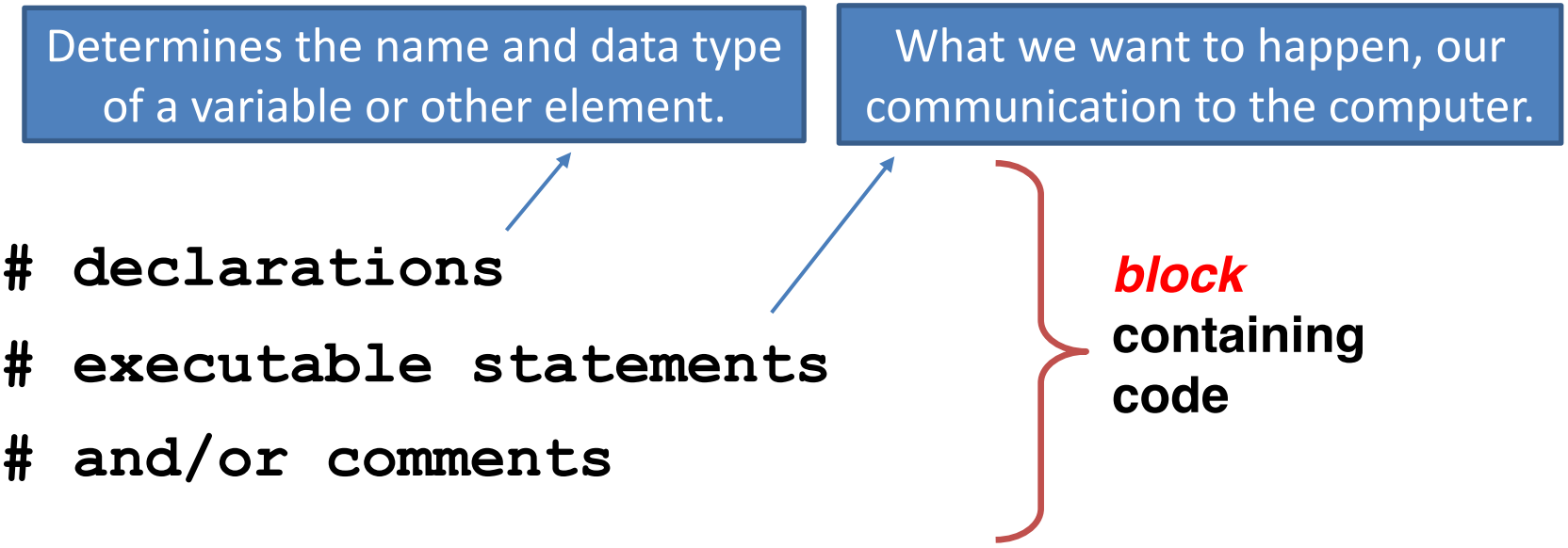
your loving son Mikey.

- Words must be in the correct order to create meaningful sentences.
  - You could completely change the meaning of a letter if the paragraph order (or even sentence or word order) was changed.
  - Still syntactically correct, but semantically quite different.
    e.g. The cat bit the dog; the dog bit the cat.

Components of a program

- Identifiers – naming memory locations
  - Rules — usually letters, numbers, and underscore (_) only and **cannot** be the same as *reserved words in the programming language.*
  - Conventions — start with a letter (examples later)
- Comments
  - Parts of source code that are not translated – they are ignored. Included to help programmers determine:
    - Who wrote the code, when they wrote it, what the code does, the version of the code, helpful hints etc
- White Space (space, tab, blank lines)
  - Improves readability of the source code for programmers. Too little or too much – program is hard to read/understand.
- Statements – some action to carry out (can be simple or complex, made of expressions)
- Blocks – A grouping of statements (sometimes under the control of a "parent construct")
- Reserved words / keywords – defined by the programming language

Structure of a Program



- Some languages
  - Use syntax to collect statements into blocks, e.g. braces **{}**

    but **Python** just uses alignment (indentation)
  - Use syntax to separate statements, e.g. the semicolon **;**

    but **Python** can just use the 'end of line' (press 'return' on the end of a statement line)
    - **Python** itself doesn't actually require declarations.

Some example Statements

Two statements:

```
print ("Hello World!")

print ("Programming is fun!")
```

Programming Style & Documentation

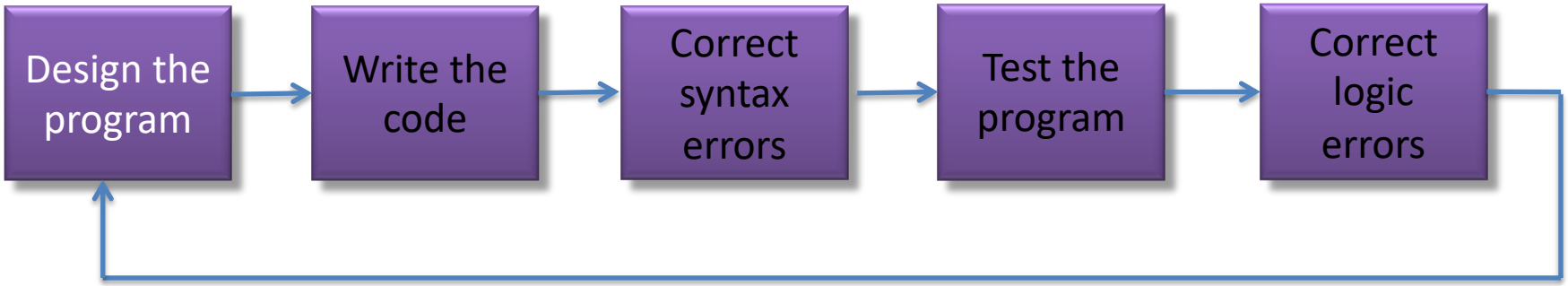
- Appropriate Comments
- Proper Indentation and Spacing

```
''' Sample program for lecture 2
Author: Son Tran
Version 2 (2021)
Demonstrates simple statements and style
'''

print ("Hello World")
print ("Programming is fun!")

# this line is a comment!
```

The Program Development Cycle



What is Python?

Developed by Guido van Rossum in 1991.

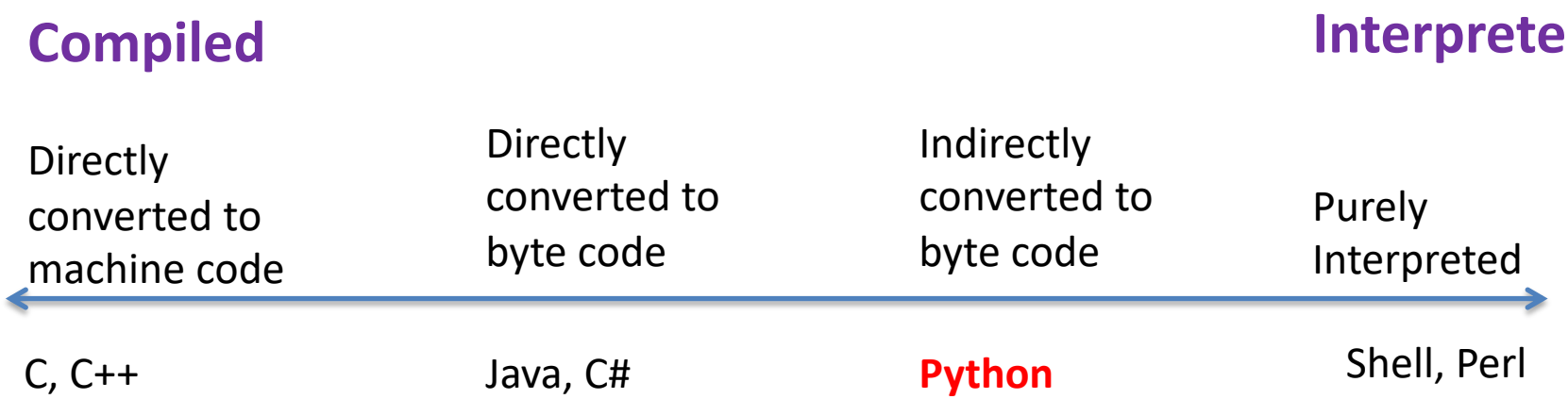
Python is a high-level general-purpose programming language that can be applied to many different classes of problems.

With Python, students can be quickly introduced to basic concepts required for successful programming.

- Easy to read
- Less development time (shorter code)
- Reduced learning time

What is Python?

Programming languages:



Who uses python?

- On-line games
- Web services
- Applications
- Science
- Instrument Control
- Embedded Systems
- Education

Python's Reserved Words (keywords)

and	else	in	try
as	except	is	while
assert	exec	lambda	with
break	finally	not	yield
class	for	or	
continue	from	pass	
def	global	~~print~~	
del	if	raise	
elif	import	return	

Ref: Meanings of python's reserved keywords:
https://www.w3schools.com/python/python_ref_keywords.asp

`print` *used to be a reserved word (in Python 2). In Python 3 now it is actually a function (a collection of blocks of statements that fulfils a task)*

When we don't follow the rules….

- If we do not follow the rules, three types of errors,

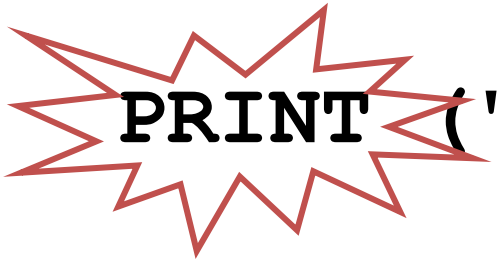
1. Problems with **syntax** (compiler errors)
   - No execution or executable program produced
   - Syntax error: prevents code from being translated

2. Problems during program execution (run-time errors)
   - e.g. divide by zero (machine can not do), wrong sort of input
   - program terminates abnormally

3. Problem with **semantics** (logical errors)
   - program runs, but produces incorrect/unexpected results

Find the error

Syntax Error – Python is case sensitive



```
PRINT ("Hello World!")
```

Find the error

```
print ("program over")

print ("program begins")
```

Semantic error - message "program over" will appear before message "program begins"

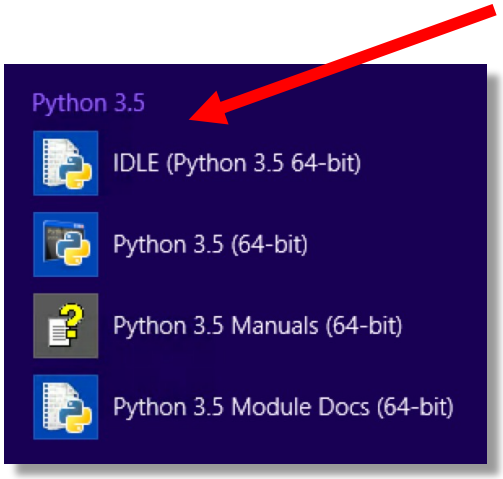
Using Python

- **Download Python:**
  It is free and accept different OS Windows, Linux/UNIX, Mac OS X, Other
  https://www.python.org/downloads/

- We want to launch the IDLE Python 3.x programming environment from the Start Menu. In ICT labs using Windows 10 you can find IDLE via:
  - Choose the Windows icon on the bottom left hand corner of your desktop
  - Move the mouse pointer and select the small arrow that appears at the bottom left of the screen
  - You'll see a list of applications installed – scroll across to the right (apps are in alphabetical order)

  - Choose **Python 3.x– IDLE (Python GUI) to start IDLE**
  - Or search for Python (we'll use this in tutorials)

The version actually installed may be 3.8.x, or 3.9.x

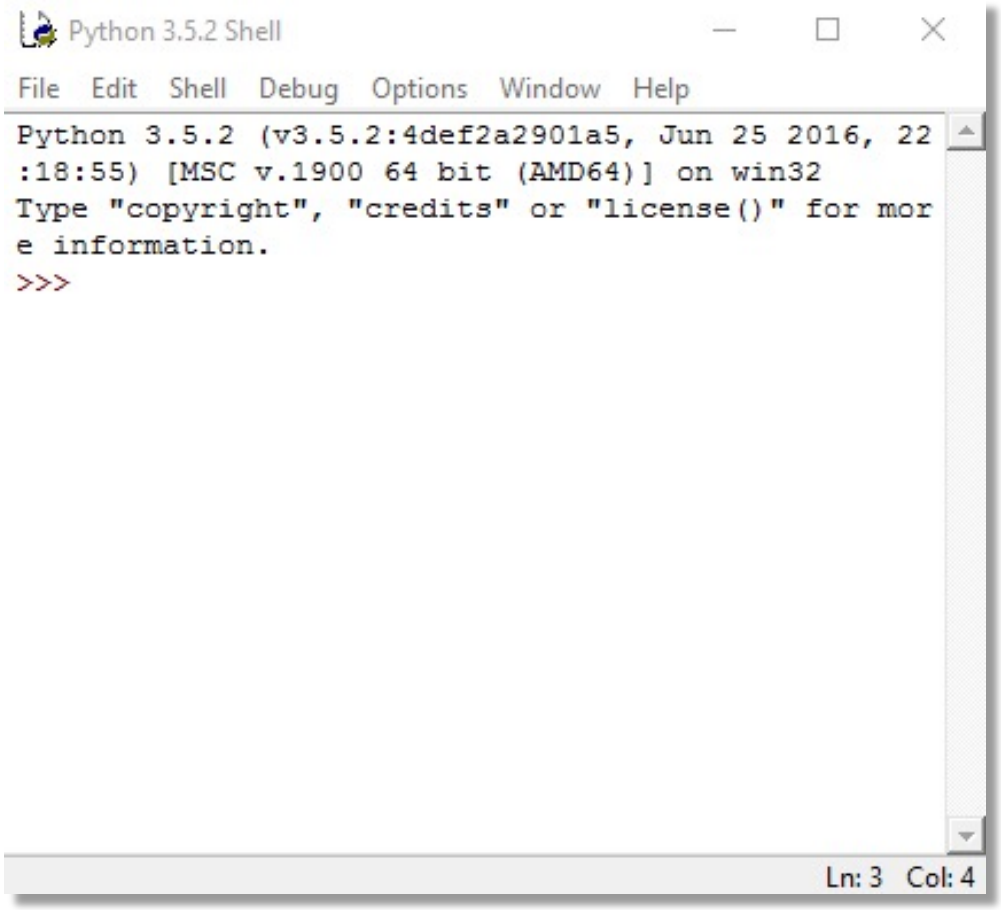


Python 2 vs Python 3

Python 3 is a newer version, but it is not totally backward compatible with Python 2. That means if you write a program using Python 2, it may not work on Python 3 and vice versa. The major (obvious) difference between the two is the **print** keyword.

- In Python 3, print is a *function*, which requires arguments (data) between parentheses e.g. `print("hello")`
- In Python 2, print is a keyword statement e.g. `print "hello"`
- (*don't worry if this is slightly confusing*)

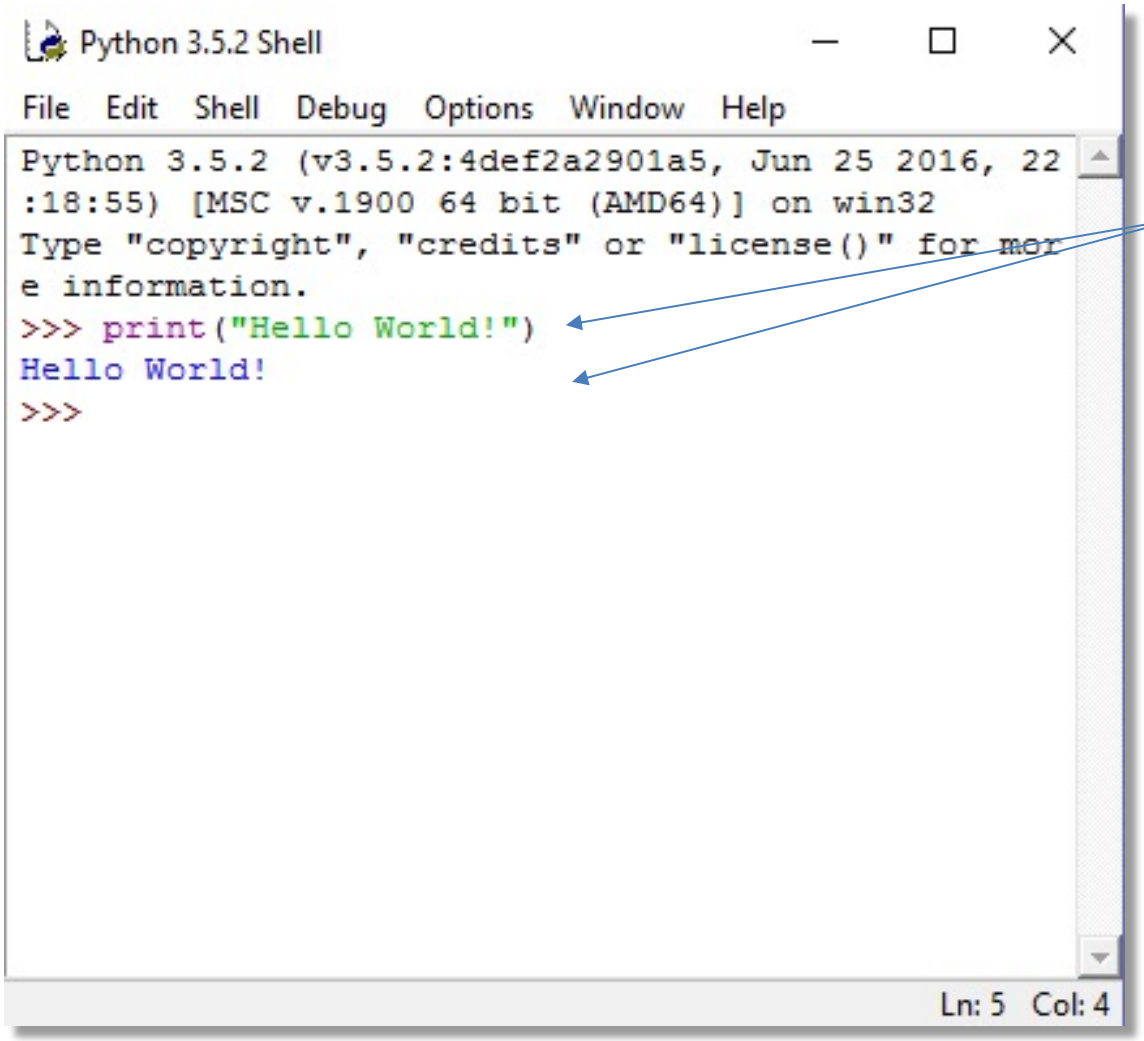
The Command Line (python interpreter)



The Console Window -> Only use for the single statement.

We almost NEVER use the console window for typing in larger sections of code.

A First Python Statement

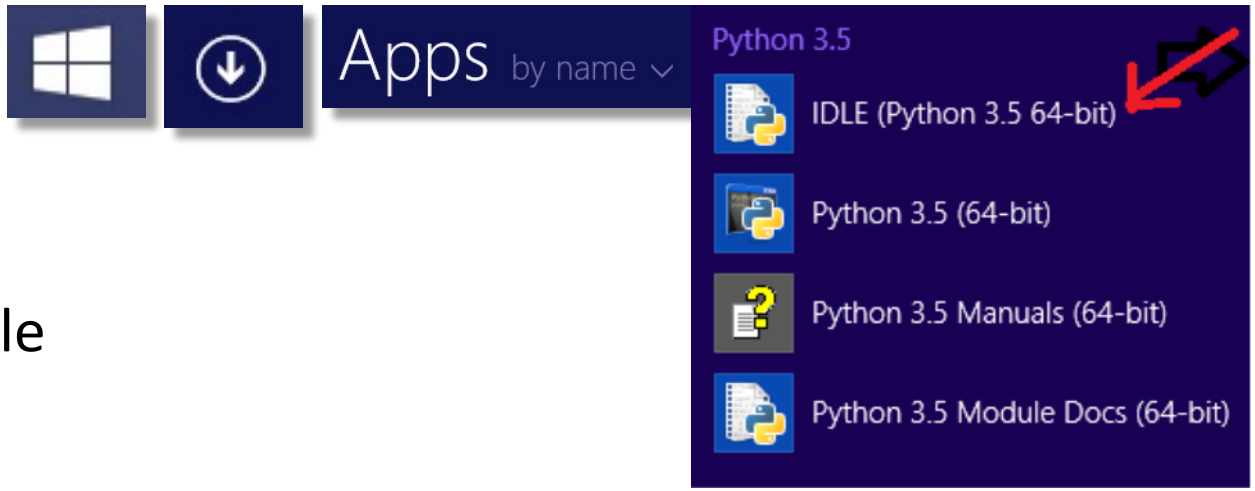


The interpreter translates and executes the statement you type in immediately

- You can create any Python application you want using just a text editor, however there are Interactive Development Environments (IDE's) that can make the process easier.
- IDLE is the IDE that comes with a Python installation.
- IDLE provides the functionality to:
  - Write Python Code
  - Perform simple and code-based editing
  - Save and Open Python (*filename.py*) files
  - Perform simple debugging tasks, and more.

Creating your first application

Start IDLE
(Windows) -

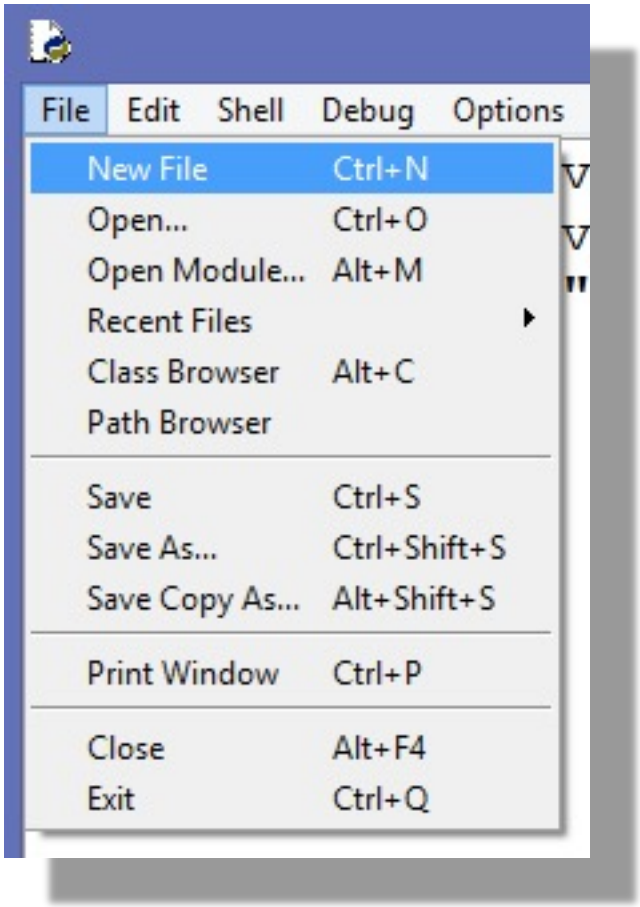


Normal workflow

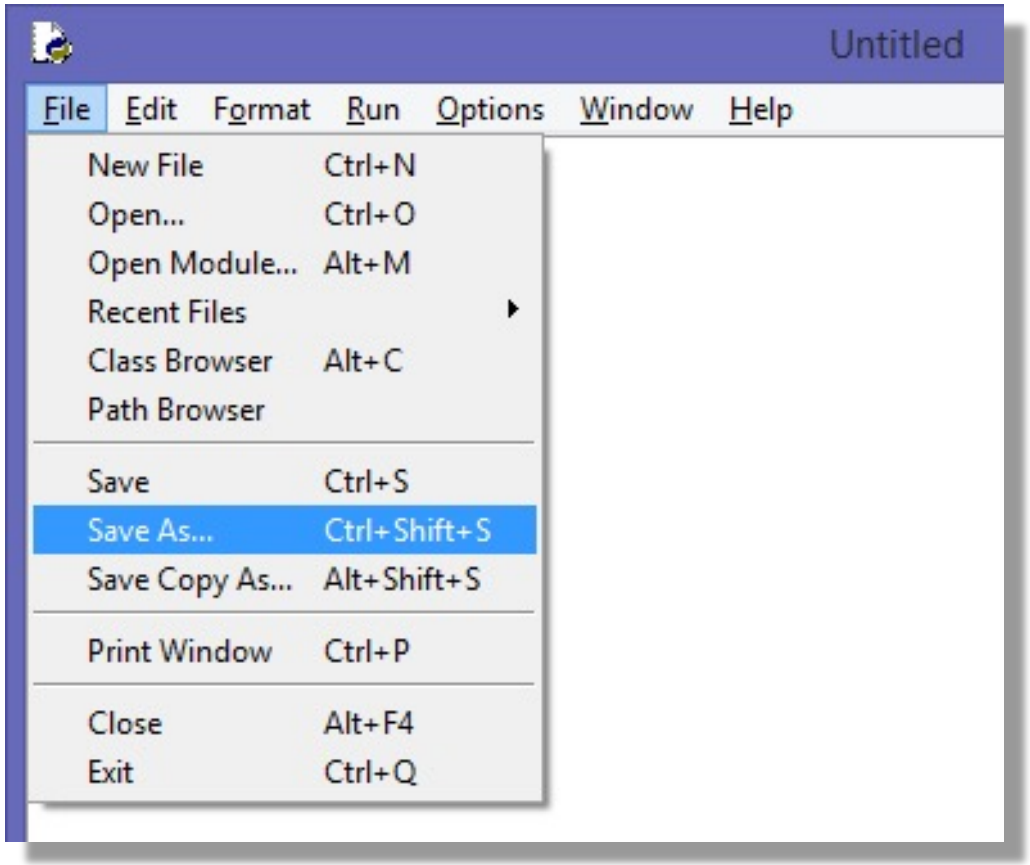
1. create a new file
2. save the file
3. edit the file
4. save the file
5. run the application from IDLE
6. go back to step 3 to fix problems

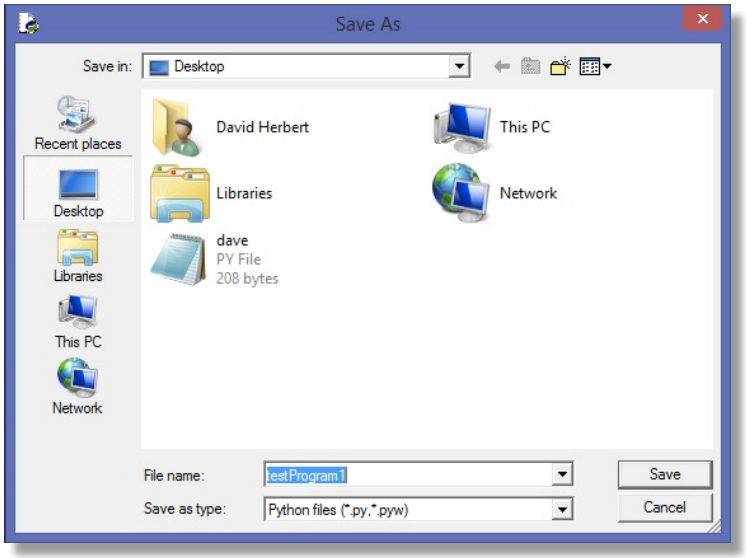
*When you run the file, the interpreter actually evaluates **all** of the source code for syntax etc, then it executes it if there are no errors. This is far more efficient than typing in code line-by-line manually in the interpreter console window.*

1. Create a file



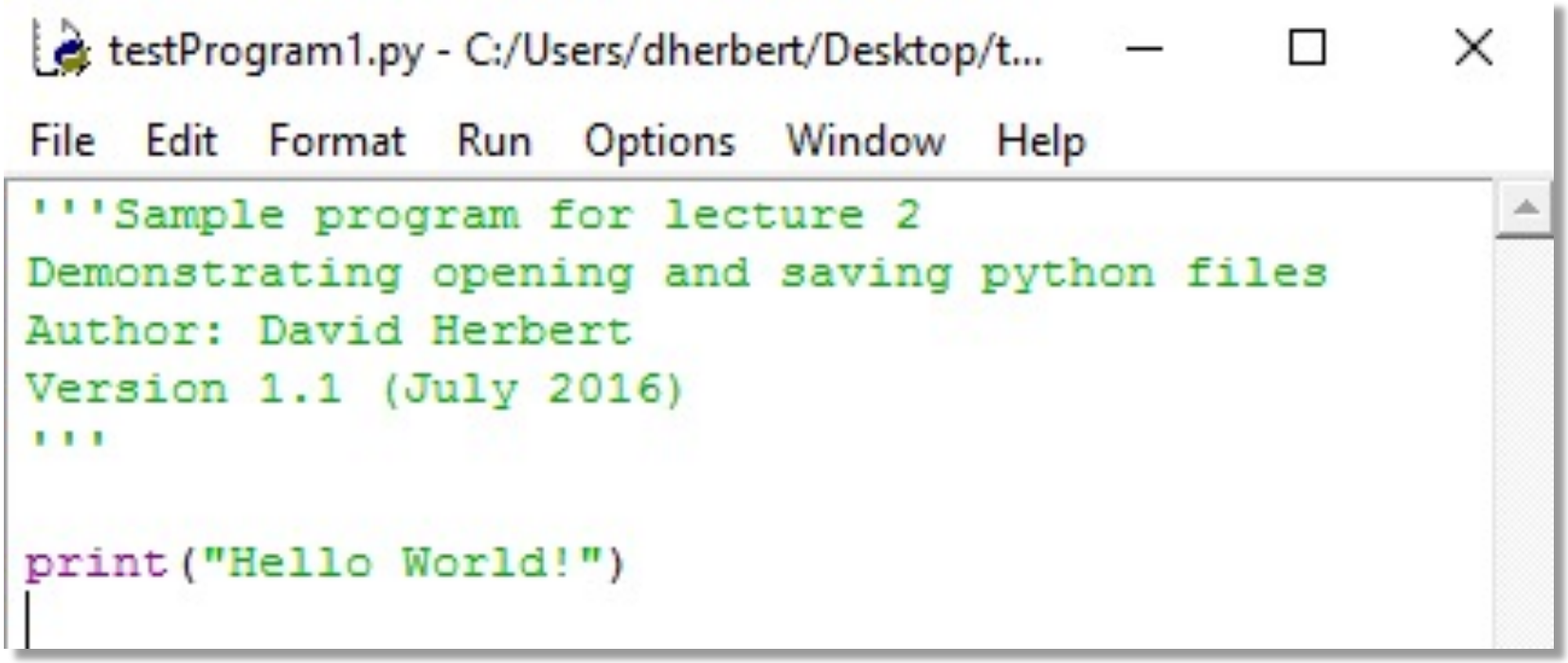
2. Save the file (Save As...)



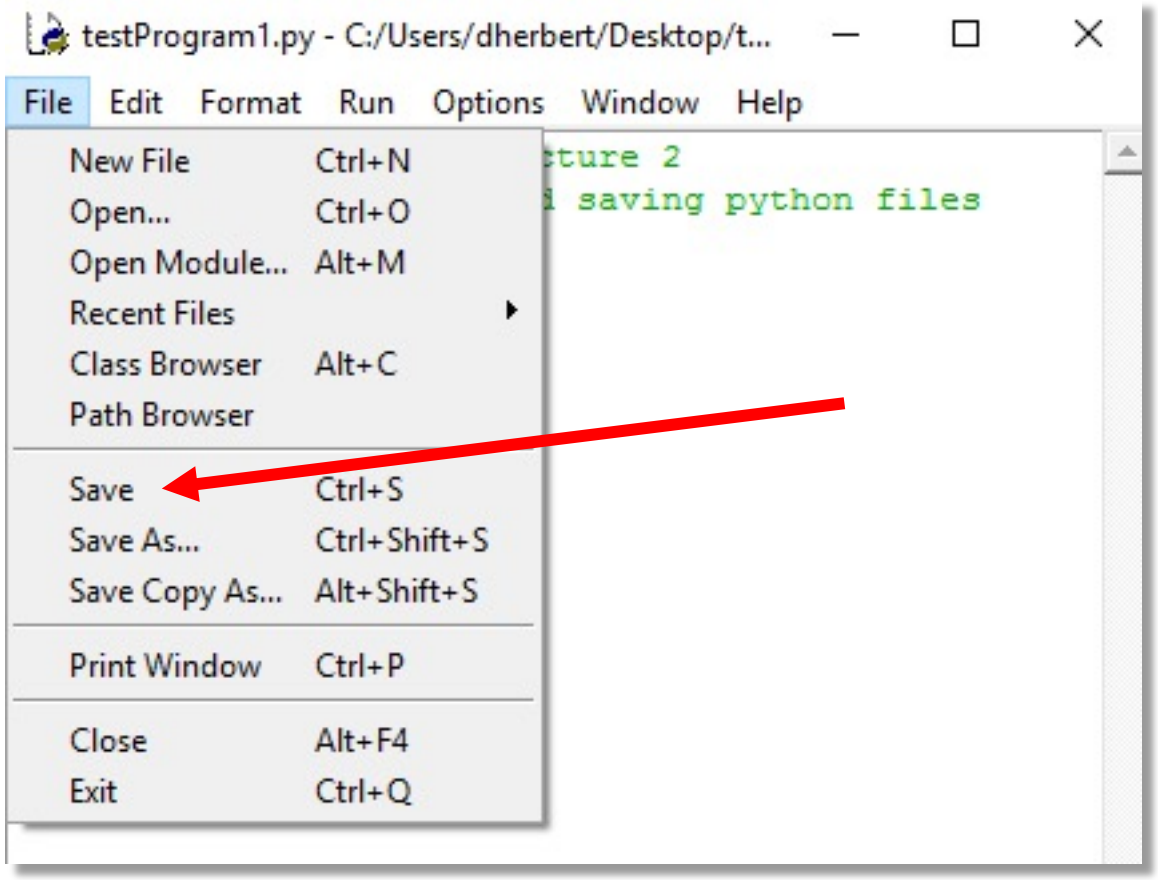


Don't forget to add a file extension of "**.py**" to your filename

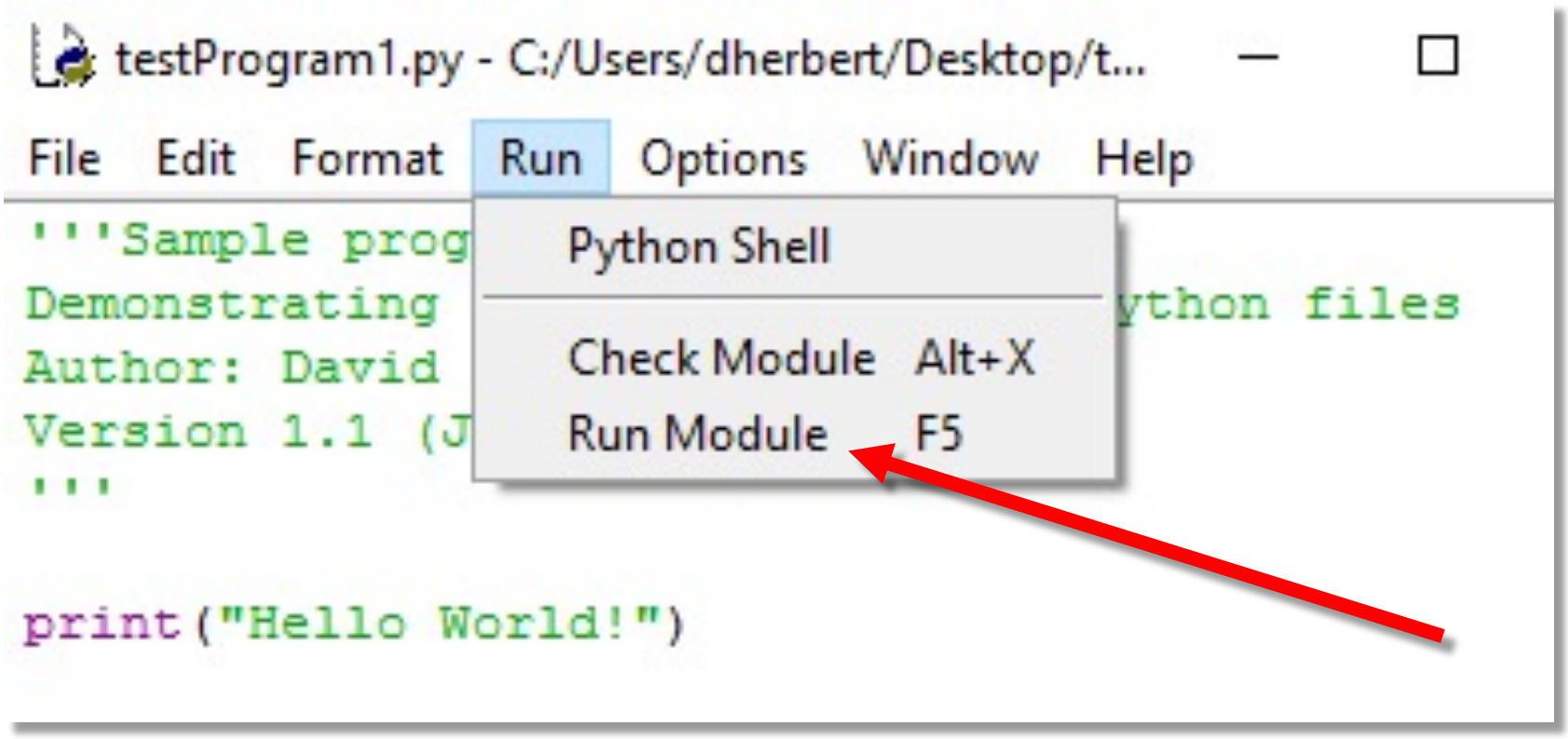
3. Edit the file



4. Save the file



5. Run the application from IDLE



When you are ready to run your program, choose **Run Module** from the **Run** menu

Python on your own device

- Python works on a number of platforms (combination of computer hardware and operating systems software).
- To get the right version visit: http://www.python.org/download/ (you will need to scroll down to the download section).
- Most Mac computers will already have Python 2 installed. You should install version 3 from python.org and use the IDLE application for this unit (unless you are comfortable using the *terminal* application and/or another IDE)

Resources

- Unit MyLO Web site
  - Lecture slides/recordings
  - Tutorial slides/recordings
  - Assessment information (Portfolio tasks, samples of test 1&2)
- Python Web sites
  - http://docs.python.org/3/tutorial/
  - http://wiki.python.org/
  - http://www.learnpython.org
- Book
  - Tony Gaddis: Starting Out With Python, Global Edition, 4th Edition (it has been uploaded in MyLO)

Questions?