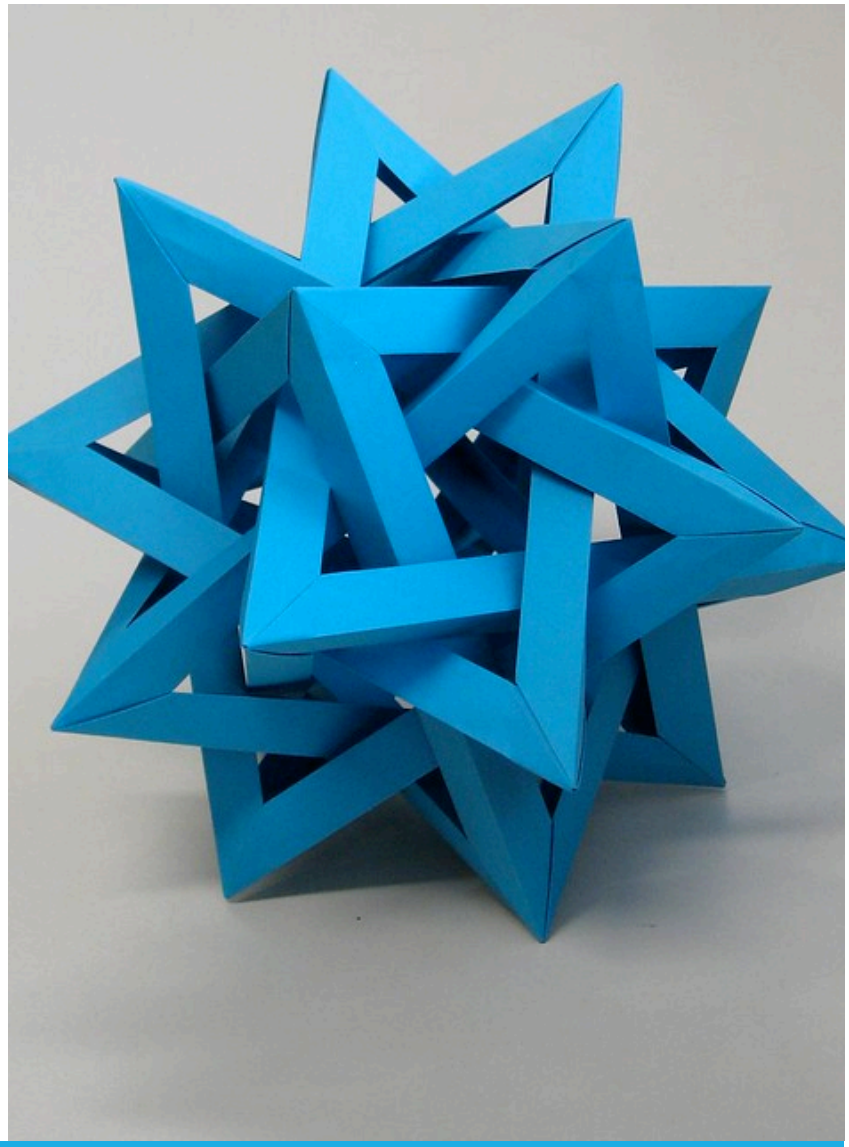
POLITECNICO
DI TORINO
Dipartimento
di Automatica e Informatica

Unit P1: Introduction to programming

A SHORT INTRODUCTION TO HARDWARE, SOFTWARE, AND ALGORITHM DEVELOPMENT

Chapter 1



Unit P1: Goals

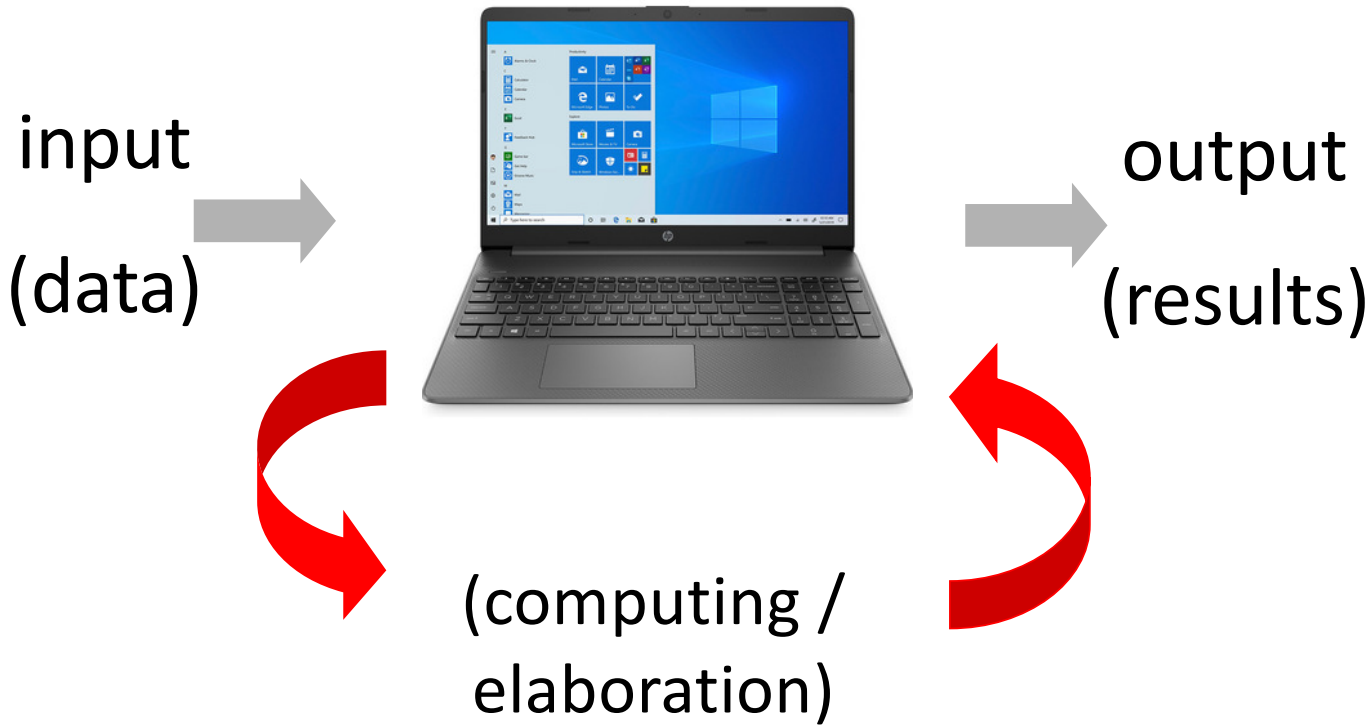
- Introduction to computers and programming
  - About computer hardware, software and programming
  - How to write and execute your first Python program
  - How to diagnose and fix programming errors
  - How to use pseudocode to describe an algorithm
- Flow Charts as a support for problem solving
  - Representation
  - Design steps
- Introduction to Python
  - Tools
  - Language

Introduction to Computer Systems

Definition of Computer Science

- Computer Science (Informatics) is the science devoted to the study of how to represent and manipulate **information**

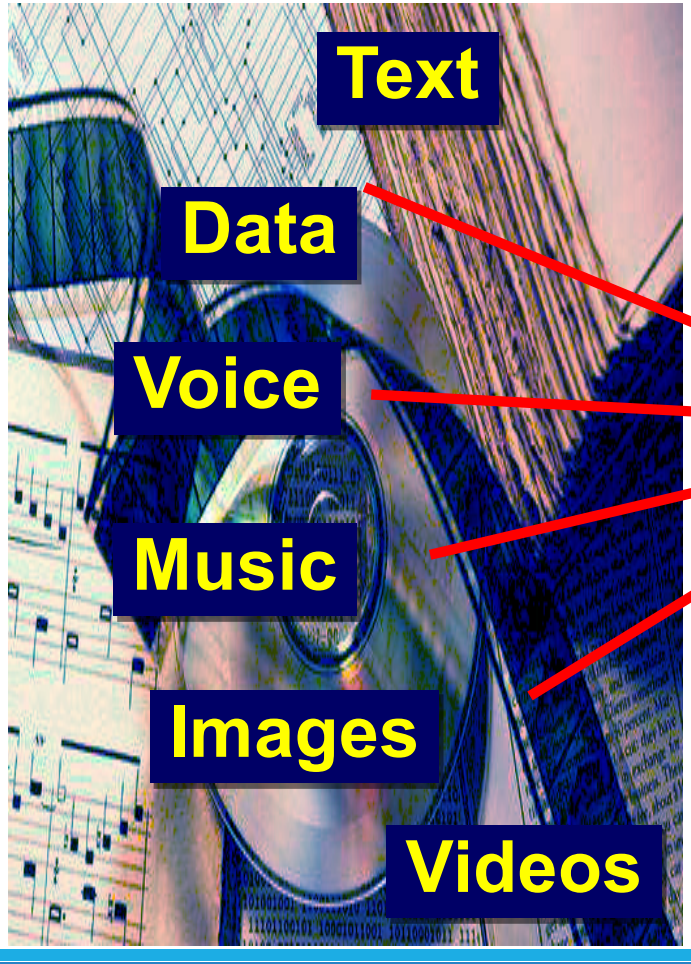
Electronic Computer



Problems

- How to encode data in a format that can be understood by the computer
- How to encode the orders into a sequence of operations that composes the desired elaboration
- How to decode the results in a format that can be understood by the human user

Digital information: everything becomes “bits”



01101100

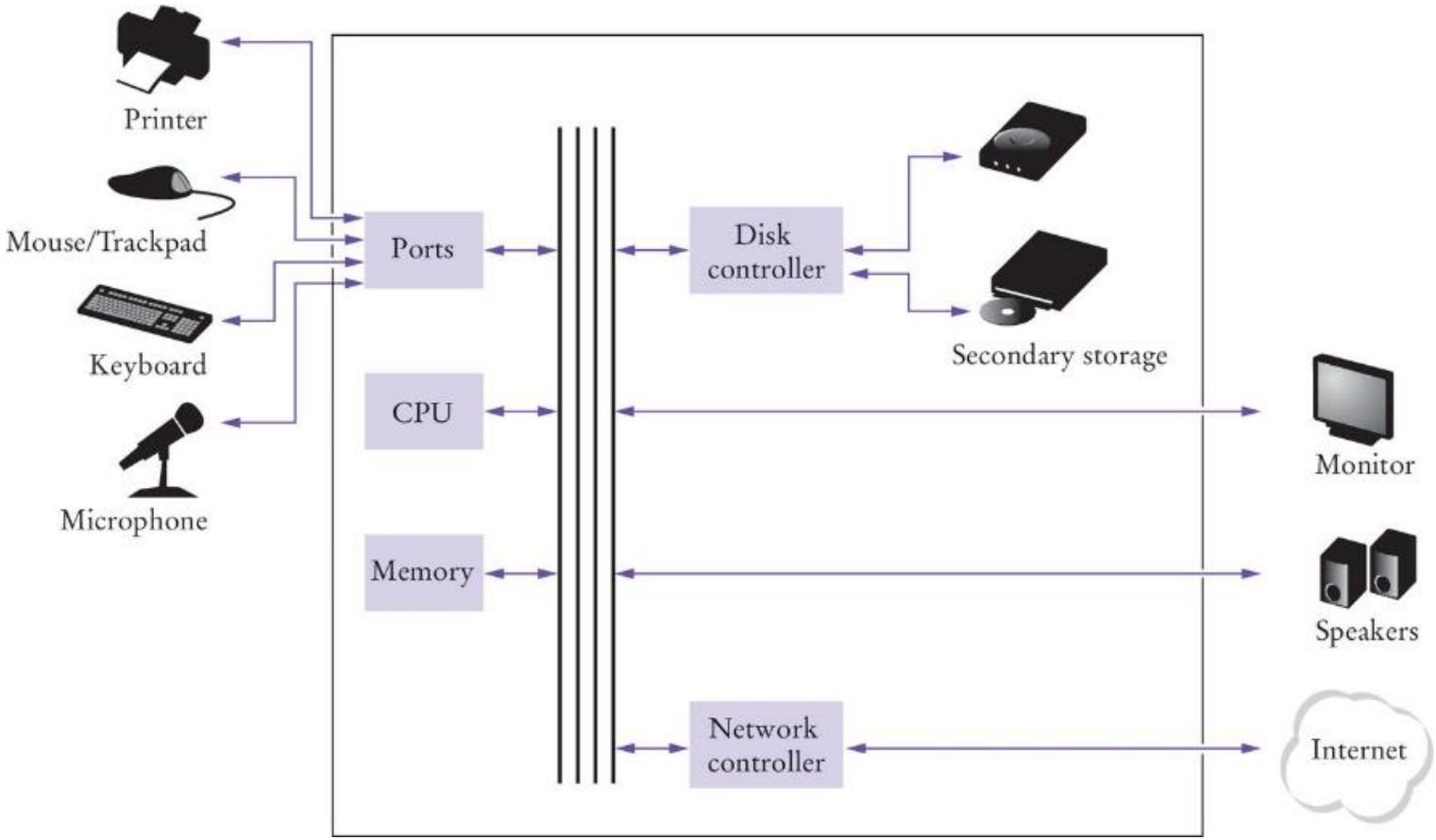
Hardware and Software

- An electronic computer is composed of two parts:
  - Hardware: Physical component, consisting of electronic devices, and mechanical, magnetic and optical parts
  - Software: “intangible” component consisting of:
    - Programs: The “instructions” for hardware
    - Data: Information upon which programs operate

Hardware

- **Hardware** consists of the physical elements in a computer system.
  - Examples: monitor, mouse, external storage, keyboard, ….
- The **central processing unit** (CPU) performs program control and data processing
- **Storage devices** include memory (RAM) and secondary storage
  - Hard disk
  - Flash drives
  - CD/DVD drives
- **Input / output devices** allow the user to interact with the computer
  - Mouse, keyboard, printer, screen…

Simplified View of a Computer's Hardware



Software

- **Software** is typically realized as an application program
  - Microsoft Word is an example of software
  - Computer Games are software
  - Operating systems and device drivers are also software
- Software
  - Software is a sequence of instructions and decisions implemented in some language and translated to a form that can be executed or run on the computer.
  - Software operates an manages the data used by the instructions
- Computers execute very basic instructions in rapid succession
  - The basic instructions can be grouped together to perform complex tasks
- Programming is the act of designing and implementing computer programs

Computer Programs

- A computer program tells a computer the sequence of steps needed to complete a specific task
  - The program consists of a (very) large number of primitive (simple) instructions
- Computers can carry out a wide range of tasks because they can execute different programs
  - Each program is designed to direct the computer to work on a specific task
- **Programming:**
  - The act (and the **art**) of designing, implementing, and testing computer programs

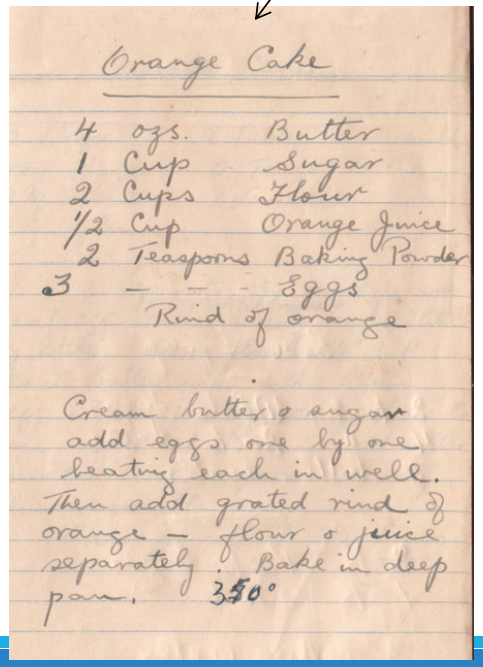
Executing a Program

- Program instructions and data (such as text, numbers, audio, or video) are stored in digital format
- To execute a program, it must be brought into memory, where the CPU can read it.
- The CPU runs the program one instruction at a time.
  - The program may react to user input.
- The instructions and the user input guide the program execution
  - The CPU reads data (including user input), modifies it, and writes it back to memory, the screen, or secondary storage.

Cooking vs Programming

Hardware

Software

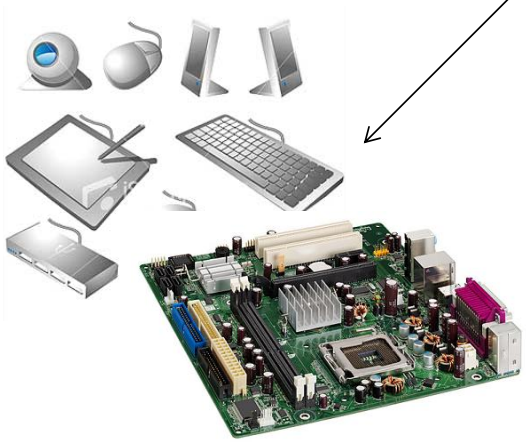

Orange Cake

4 ozs. Butter
1 Cup Sugar
2 Cups Flour
½ Cup Orange Juice
2 Teaspoons Baking Powder
3 — — — Eggs
Rind of orange

Cream butter & sugar add eggs one by one beating each in well. Then add grated rind of orange — flour & juice separately. Bake in deep pan. 350°

Cooking vs Programming

Hardware



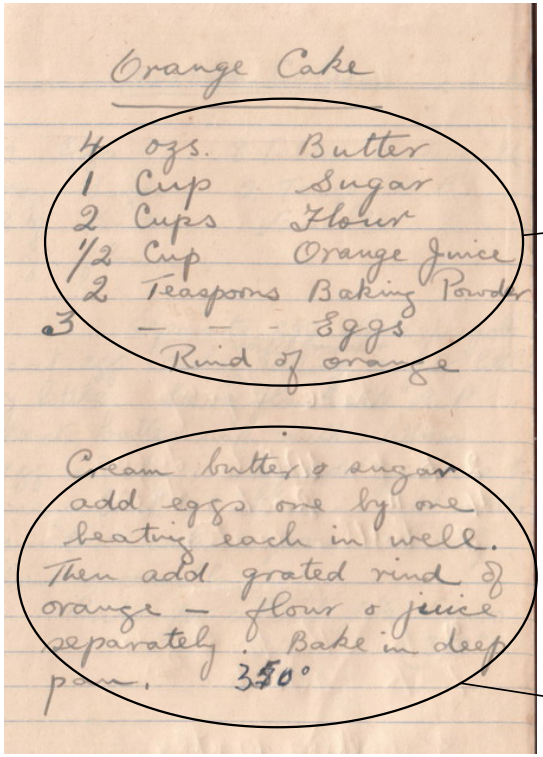
Software

```
void CruiseControl_init(_C_CruiseControl * _C_)
{
    CruiseSpeedMgt_init(&(_C_->_C0_CruiseSpeedMgt));
    CruiseStateMgt_init(&(_C_->_C3_CruiseStateMgt));
    (_C_->_M_condact_0) = true;
    ThrottleCmd_init(&(_C_->_C4_ThrottleCmd));
    (_C_->_M_init) = true;
}

/* ========= */
/* MAIN NODE */
/* ========= */

void CruiseControl(_C_CruiseControl * _C_)
{
    bool BrakePressed;
    bool AcceleratorPressed;
    bool SpeedOutOffLimits;
    bool _L19;
/*#code for node CruiseControl */
/* call to node not expanded DetectPedalsPressed */
    (_C_->_Cn_DetectPedalsPressed._I0_Brake) = (_C_->_I
    (_C_->_Cn_DetectPedalsPressed._I1_Accelerator) = (
    DetectPedalsPressed(&(_C_->_Cn_DetectPedalsPressed)
    BrakePressed = (_C_->_Cn_DetectPedalsPressed._O0_Br
    AcceleratorPressed =
        (_C_->_Cn_DetectPedalsPressed._O1_AcceleratorPr
/* call to node not expanded DetectSpeedLimits */
    (_C_->_Cn_DetectSpeedLimits._I0_speed) = (_C_->_I8_
    DetectSpeedLimits(&(_C_->_Cn_DetectSpeedLimits));
    SpeedOutOffLimits = (_C_->_Cn_DetectSpeedLimits._O0_
/* call to node not expanded CruiseStateMgt */
    (_C_->_C3_CruiseStateMgt._I0_BrakePressed) = BrakeP
```

Cooking vs Programming



Ingredients/Data

```
/* ========= */
/* MAIN NODE */
/* ========= */

void CruiseControl(_C_CruiseControl * _C_)
{
    bool BrakePressed;
    bool AcceleratorPressed;
    bool SpeedOutOffLimits;
    bool _L19;
/*#code for node CruiseControl */
/* call to node not expanded DetectPedalsPressed */
    (_C_->_Cn_DetectPedalsPressed._I0_Brake) = (_C_->_I
    (_C_->_Cn_DetectPedalsPressed._I1_Accelerator) = (_
    DetectPedalsPressed(&(_C_->_Cn_DetectPedalsPressed)
    BrakePressed = (_C_->_Cn_DetectPedalsPressed._O0_Br
    AcceleratorPressed =
        (_C_->_Cn_DetectPedalsPressed._O1_AcceleratorPr
/* call to node not expanded DetectSpeedLimits */
    (_C_->_Cn_DetectSpeedLimits._I0_speed) = (_C_->_I8_
    DetectSpeedLimits(&(_C_->_Cn_DetectSpeedLimits));
    SpeedOutOffLimits = (_C_->_Cn_DetectSpeedLimits._O0_
/* call to node not expanded CruiseStateMgt */
    (_C_->_C8_CruiseStateMgt._I0_BrakePressed) = BrakeP
```

Instructions/Code

What is “programming”?

- Programming amount to writing a “document” (source file) that describes the solution to the problem at hand

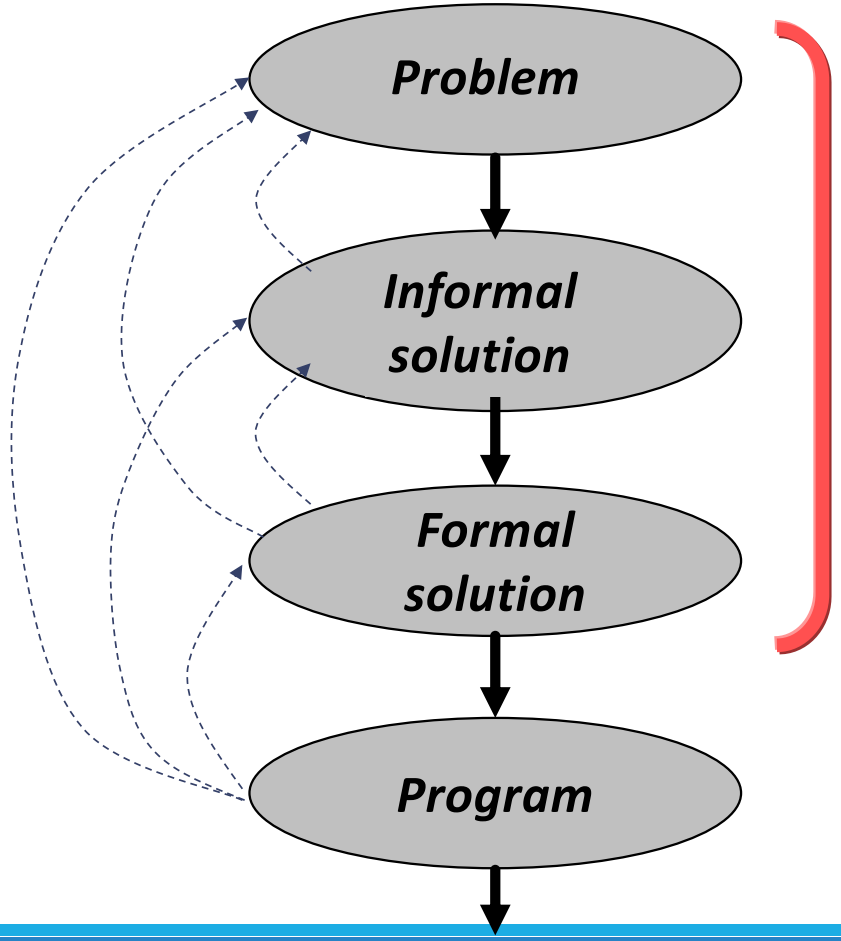
- In general, “the” solution to a given problem does not exist
  - Programming consists of finding the most efficient solution to the problem, according to certain metrics

What is “programming”?

- Programming is a “**creative**” task!
  - Every problem is different from every other problem
  - There are no systematic/analytic solutions or “universal” solutions!

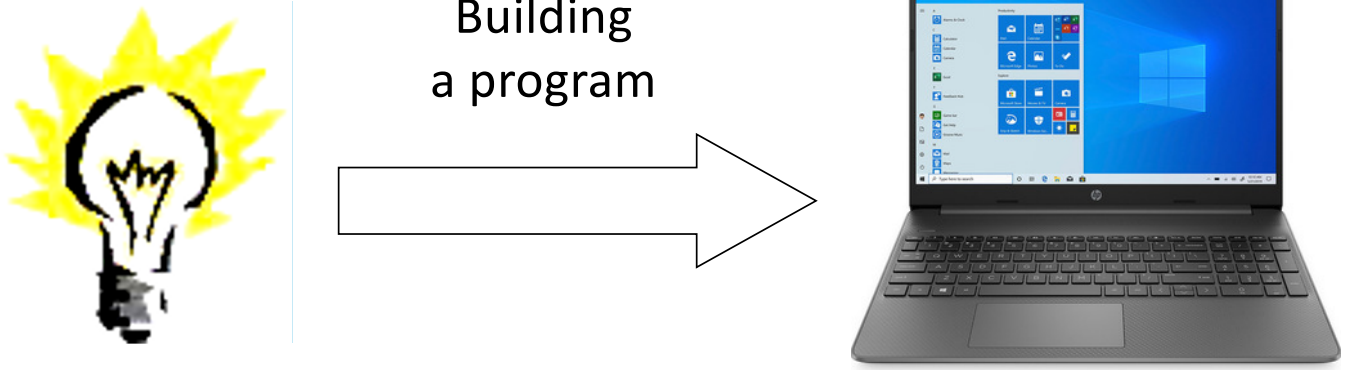
- Programming is a complex operation
  - A “direct” approach (from the problem directly to the final source code) is unfeasible
  - Usually, organized in several stages (refinements)

Developing a program



What do we learn in this course?

- From the specification of a problem, to the realization of a solution to that problem, in the form of a computer program



The first ENGINEERING DESIGN course at Politecnico

What will we learn in this course?

- To acquire the mental attitude (*forma mentis*) necessary to face «problem posing» and «problem solving» tasks
  - To analyze a problem, and decompose it in smaller problems
  - To describe the solution of a problem, in clear non-ambiguous way
  - To analyze and make explicit our reasoning steps
- This is useful in all scientific (and non-scientific) disciplines
- In particular…
  - …to **THINK** like a computer
  - …to **SPEAK** to a computer so that it can understand us

Tools for Problem Solving

The Rules

- I must solve the problem, and imagine I have one sheet of paper (memory) and a pen, only
- Every information that I need to remember must be written on the paper

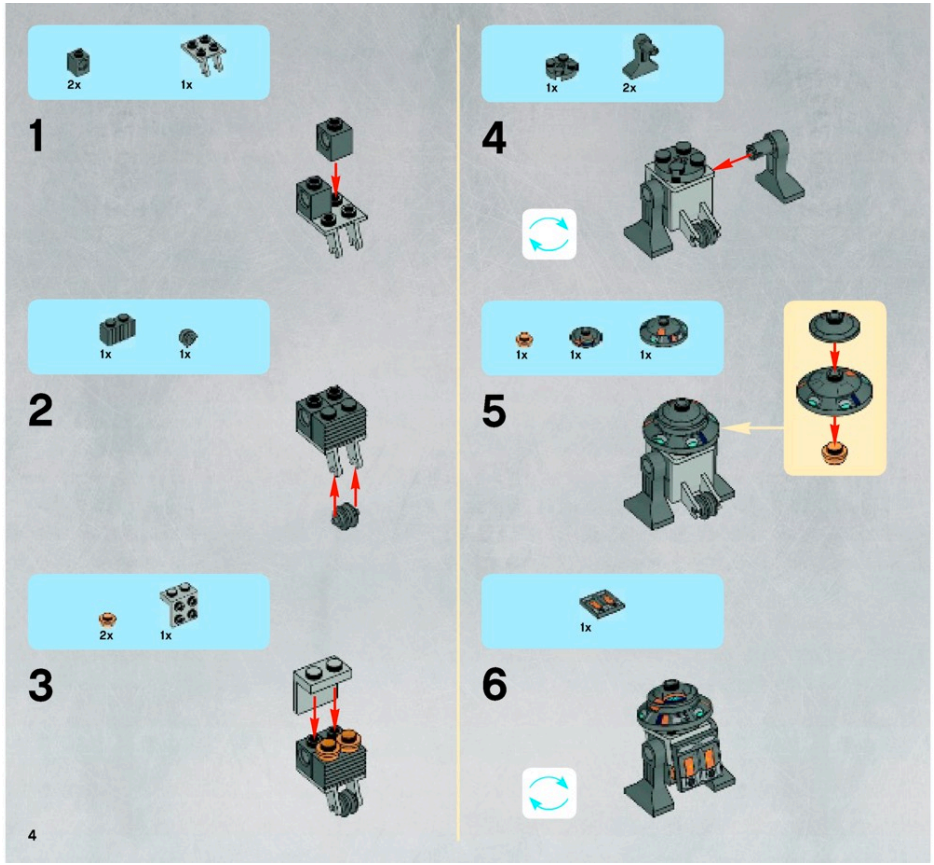
Exercise

- Find who is the oldest student in the classroom

Algorithms

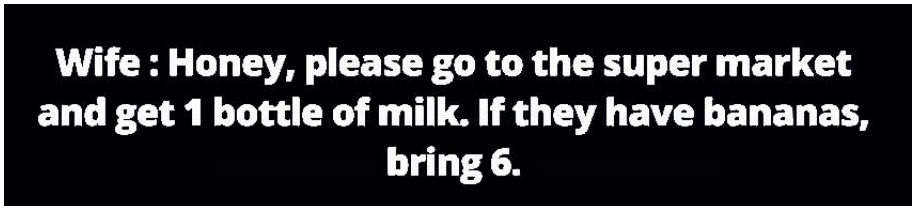
1.7

Algorithm

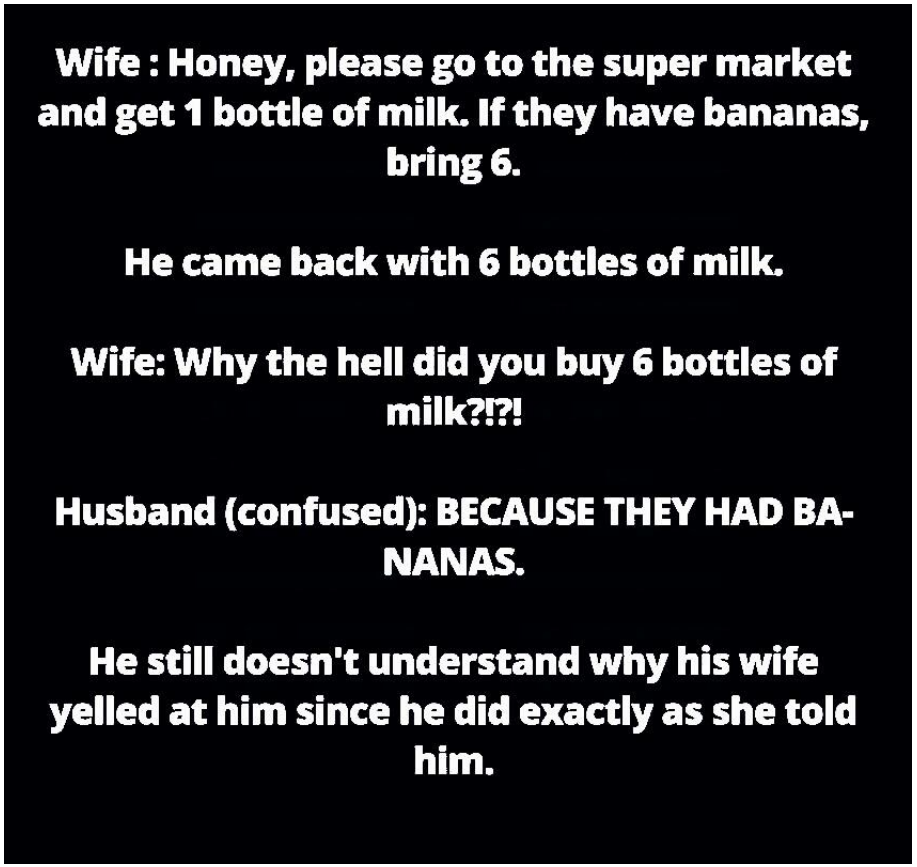


- Structured visual language
- Sequential operations
- Sub-operations
- Repetitions

Thinking like a computer programmer ☹



Thinking like a computer programmer ☹



Our First Definition

- **Algorithm**
- An algorithm is a step by step description of how to solve a problem

Introduction to Algorithms

- If you want a computer to perform a task, you start by writing an algorithm
- An **Algorithm** is:
  - A sequence (the order matters!) of actions to take (instructions) to accomplish the given task for achieving a goal
  - Like a ‘recipe’
- For complex problems, software developers study an algorithm, then formalize it with pseudo code or flow charts, all before attempting to write a computer program
- Developing algorithms is a fundamental **problem solving** skill
  - It has uses in many fields outside of Computer Science

Algorithm: Formal Definition

- An algorithm describes a sequence of steps that is:
- Unambiguous
  - No “assumptions” are required to execute the algorithm
  - The algorithm uses precise instructions
- Executable
  - The algorithm can be carried out in practice
- Terminating
  - The algorithm will eventually come to an end, or halt

Problem Solving: Algorithm Design

- Algorithms are simply plans
  - Detailed plans that describe the steps to solve a specific problem
- You already know quite a few
  - Calculate the area of a circle
  - Find the length of the hypotenuse of a triangle
- Some problems are more complex and require more steps
  - Solve a 2^{nd} degree equation
  - Find the GCD of two numbers
  - Calculate PI to 100 decimal places
  - Calculate the trajectory of a missile

Algorithms in everyday life

1. Pour some water
2. Light the fire
3. Wait
4. If water is not boiling, go back to 3
5. Add pasta
6. Wait a little
7. Taste
8. If it's still raw go back to 6
9. Remove water

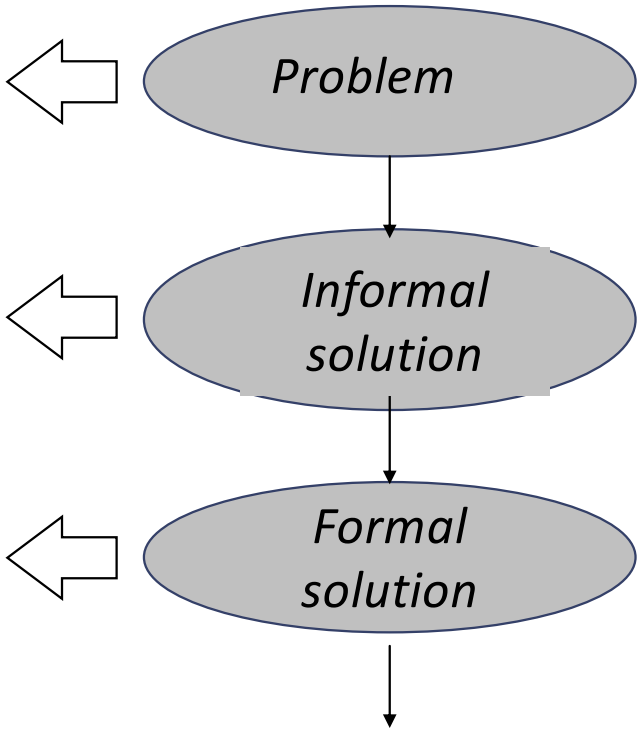
Who noticed that we forgot the salt?

A Simple Example

- A simple algorithm to get yourself a drink of orange juice
  - For simplicity, the following are true:
    - You have a clean glass in the cabinet
    - You have orange juice in your refrigerator
- So one valid algorithm is:
  - get a glass from your cabinet
  - go to the refrigerator and get the orange juice container
  - open the orange juice container
  - pour the orange juice from the container into the glass
  - put the orange juice container back in the refrigerator
  - drink your juice

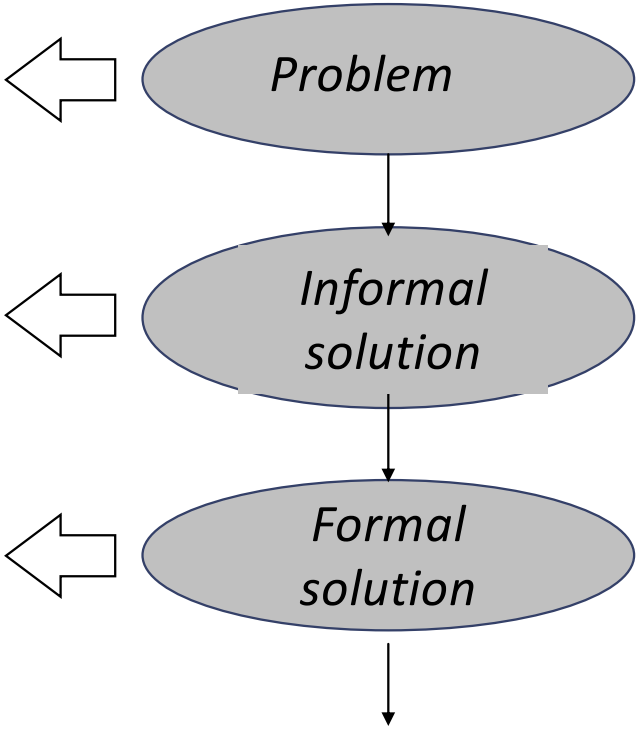
Example of problem solving

- Problem: Compute the maximum among A and B
- Solution: The maximum is the larger number between A and B…
- Formal solution:
  1. initially: max = 0
  2. if A > B then max = A; stop
  3. otherwise max = B; stop



Example of problem solving

- Problem: Compute the Greatest Common Divisor (GCD) among A and B
- Solution: By definition, the GCD is the largest integer number that divides exactly (without remainder) both A and B. Let's try all the numbers from 1 to A or B.
- Formal solution: ???



Second Example: Selecting a Car

- Problem Statement:
  - You have the choice of buying two cars.
  - One is more fuel efficient than the other, but also more expensive.
  - You know the price and fuel efficiency (in miles per gallon, mpg) of both cars.
  - You plan to keep the car for ten years.
  - Which car is the better deal?

Developing the Algorithm

- Determine the inputs and outputs
- From the problem statement we know:
  - Car 1: Purchase price, Fuel Efficiency
  - Car 2: Purchase price, Fuel Efficiency
  - Price per gallon = $4.00
  - Annual miles driven= 15,000
  - Length of time = 10 years
- For each car we need to calculate:
  - Annual fuel consumed for each car
  - Annual fuel cost for each car
  - Operating cost for each car
  - Total cost of each Car
- Then we select the car with the lowest total cost

Formalizing the solution

- Pseudo-code
  - Pros
    - Immediate
  - Cons
    - Algorithm description not very abstract
    - More complex interpretation
- Flow Charts
  - Pros
    - More intuitive: graphical formalism
    - More abstract algorithm description
  - Cons
    - Require learning the meaning of different blocks
    - Struggle to represent complex or more abstract operations

Translating the Algorithm to pseudocode

- Break down the problem into smaller tasks
  - ‘Calculate total cost’ for each car
  - To calculate the total cost for each year we need to calculate the operating cost
  - The operating cost depends on the annual fuel cost
  - The annual fuel cost is the price per gallon * the annual fuel consumed
  - The annual fuel consumed is the annual miles drive / fuel efficiency
- Describe each subtask as pseudocode
  - total cost = purchase price + operating cost

The Pseudo-code

- For each Car, compute the total cost
  - Annual fuel consumed = annual miles driven / fuel efficiency
  - Annual fuel cost = price per gallon * annual fuel consumed
  - Operating cost = Length of time * annual fuel cost
  - Total cost = purchase price + operating cost
- If total cost1 < total cost2
  - Chose Car1
- Else
  - Choose Car2

Bank Account Example

- Problem Statement:
  - You put $10,000 into a bank account that earns 5 percent interest per year. How many years does it take for the account balance to be double the original?
- How would you solve it?
  - Manual method
    - Make a table
    - Add lines until done
  - Use a spreadsheet!
    - Write a formula
    - Per line, based on the line above

year	balance
0	10000
1	10000.00 x 1.05 = 10500.00
2	10500.00 x 1.05 = 11025.00
3	11025.00 x 1.05 = 11576.25
4	11576.25 x 1.05 = 12155.06

Develop the algorithm steps

- You put $10,000 into a bank account that earns 5 percent interest per year. How many years does it take for the account balance to be double the original?

year	balance
0	10000

- Break it into steps
  - Start with a year value of 0 and a balance of $10,000
  - Repeat the following while the balance is less than $20,000
    - Add 1 to the year value
    - Multiply the balance by 1.05
    - (5% increase)

year	balance
0	10000
1	10500
14	19799.32
15	20789.28

  - Report the final year value as the answer

Translate to pseudocode

- Pseudocode
  - Half-way between natural language and a programming language
- Modified Steps
  - Set the year value of 0
  - Set the balance to $10,000
  - While the balance is less than $20,000
    - Add 1 to the year value
    - Multiply the balance by 1.05
  - Report the final year value as the answer
- The pseudocode is easily translated into Python

From the Solution to the Program

- Writing a program is practically immediate, if we start from the formal solution (pseudo code or flow chart)
- Programming languages offer different constructs and statements of varying complexity
  - Depending on the language

Which languages?

- Different levels of abstraction
  - High-level languages
    - Language statements have a complexity equivalent to structured flow chart blocks (assignments, conditions, cycles, …)
    - Examples: C, C++, Java, JavaScript, Python, etc.
    - Independent from the hardware
  - Assembly languages
    - Language statements correspond to micro-architecture instructions
    - Highly dependent from the hardware
    - Example: Assembly language of the Intel Core microprocessor

Which languages? – Examples

- High Level Language

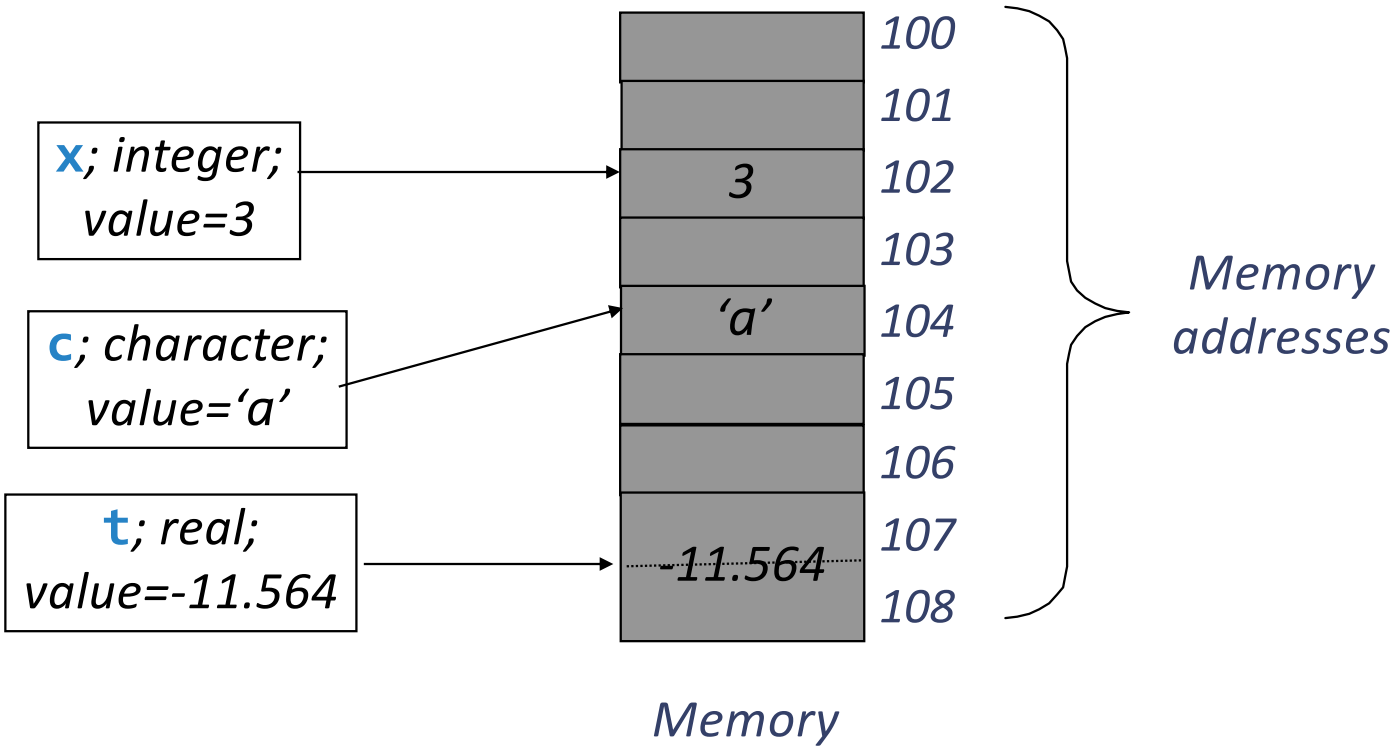
```
…
if x > 3:
  x = x + 1
…
```

- Assembly Language

```
…
LOAD Reg1, Mem[1000]
ADD Reg1, 10
…
```

Specific for a given architecture (microprocessor)

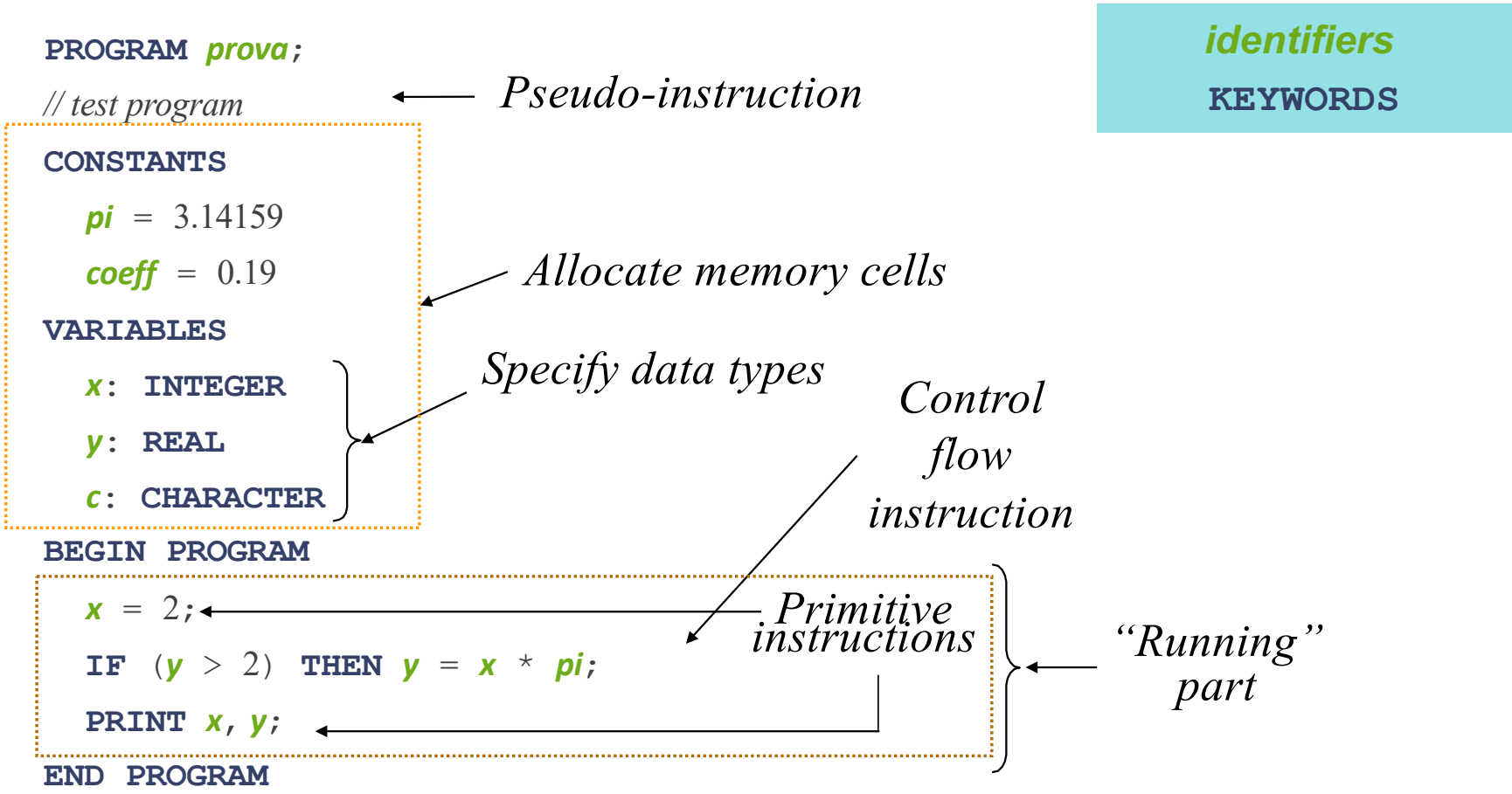
Data abstraction



Instructions

- Operations supported by the programming language, that will execute them by translating them to the machine level
  - Pseudo-instructions
    - Directive to the language interpreter or compiler, do not correspond to executable code
  - Basic (elementary, primitive) instructions
    - Operations that directly correspond to hardware operations
    - Example: interaction with I/O devices, data access and modification
  - Flow control instructions
    - Execute complex operations by controlling the execution sequence of basic instructions

Program example

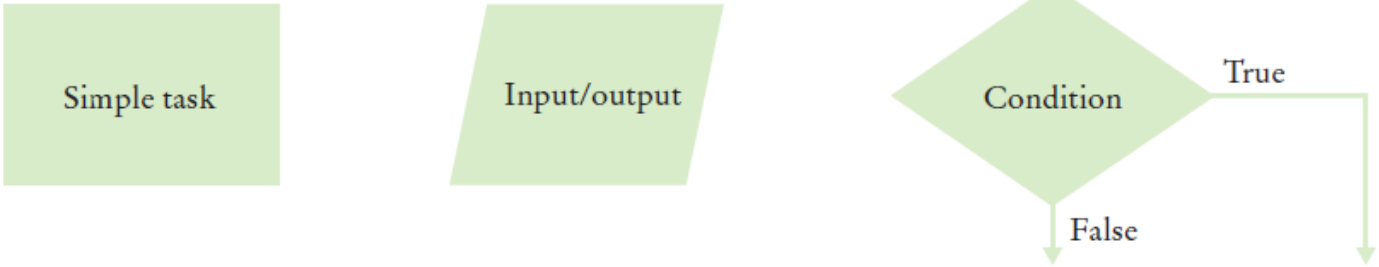


Using Flowcharts to Develop and Refine Algorithms

3.5

Problem Solving: Flowcharts

- A flowchart shows the structure of decisions and tasks to solve a problem
- Basic flowchart elements:



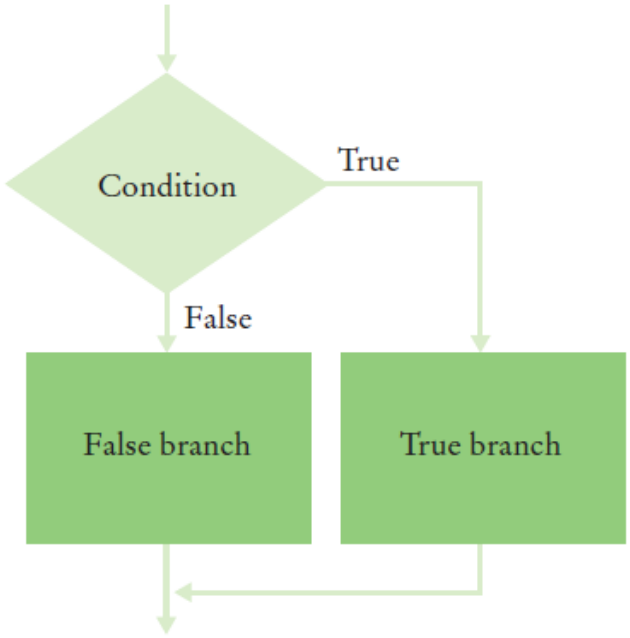
- Connect them with arrows
  - But never point an arrow inside another branch!
- Each branch of a decision can contain tasks and further decisions

Using Flowcharts

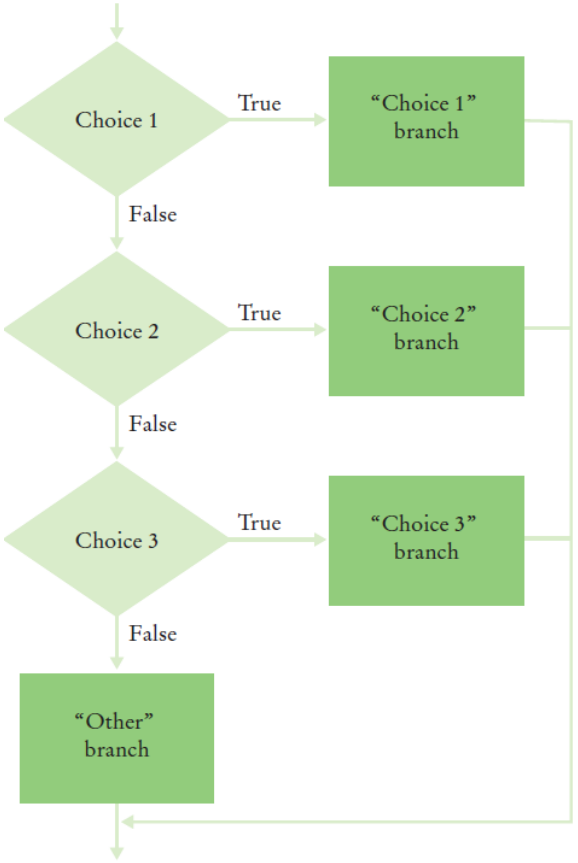
- Flowcharts are an excellent tool
- They can help you visualize the flow of your algorithm
- Building the flowchart
  - Link your tasks and input / output boxes in the sequence they need to be executed
  - When you need to make a decision use the diamond (a conditional statement) with two outcomes
  - Never point an arrow inside another branch

Conditional Flowcharts

- Two Outcomes

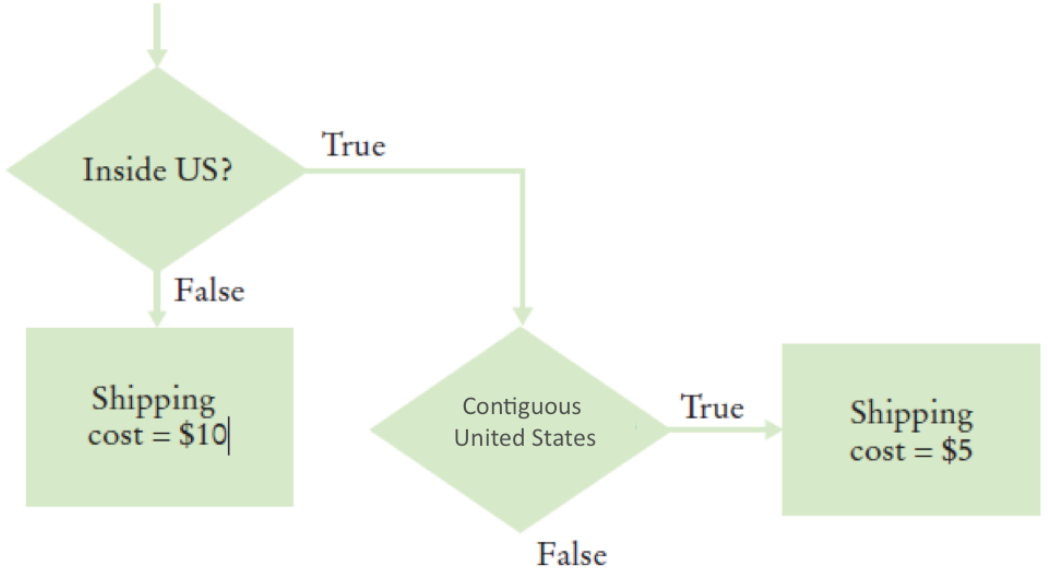


- Multiple Outcomes



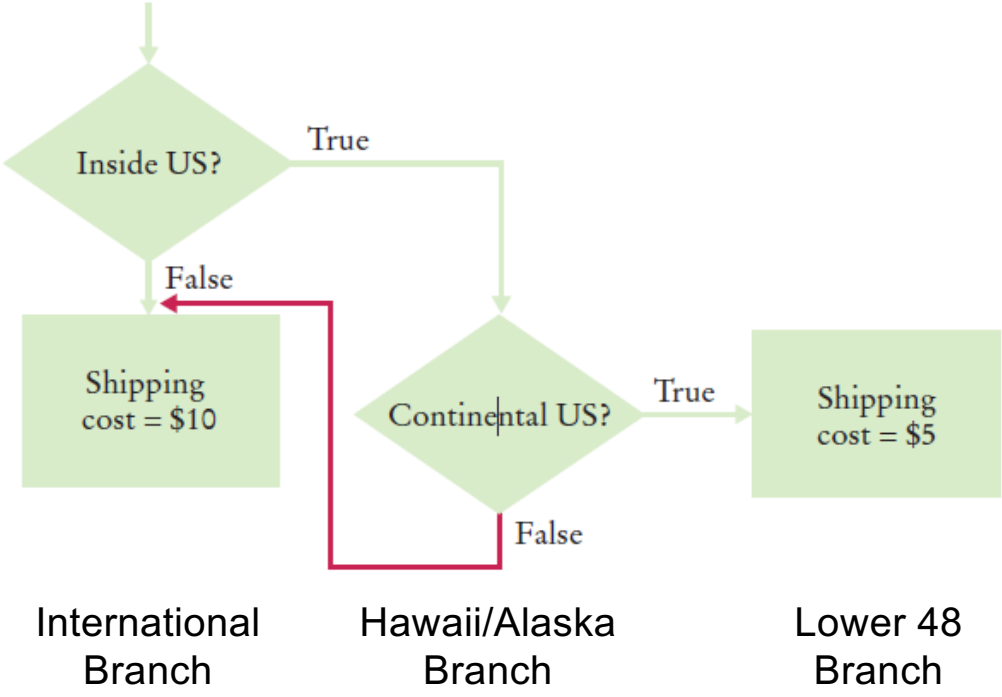
Shipping Cost flowchart

- Shipping costs are $5 inside the contiguous United States (Lower 48 states), and $10 to Hawaii and Alaska. International shipping costs are also $10.
- Three Branches:



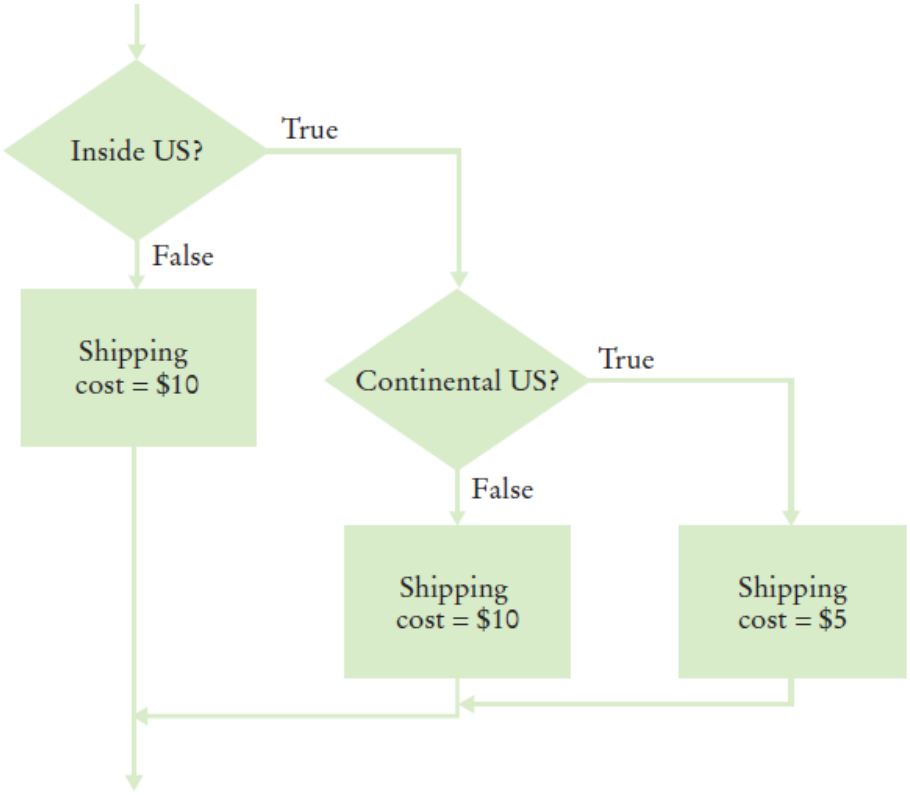
Don’t Connect Branches!

- Shipping costs are $5 inside the United States, except that to Hawaii and Alaska they are $10. International shipping costs are also $10.
- Don’t do this!



Shipping Cost Flowchart

- Shipping costs are $5 inside the United States, except that to Hawaii and Alaska they are $10. International shipping costs are also $10.



Complex Decision Making is Hard

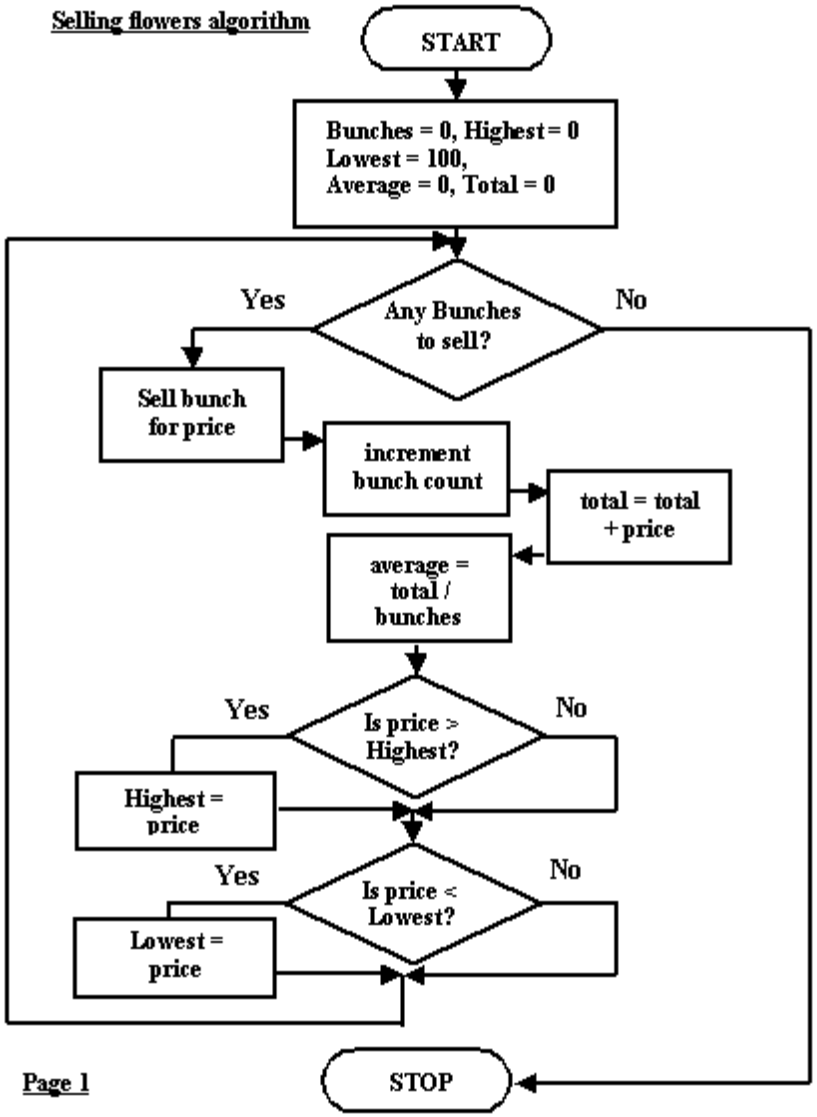
- Computer systems are used to help sort and route luggage at airports
- The systems:
  - Scan the baggage tags
  - Sorts the items
  - Routes the items to conveyor belts
  - Humans then place the bags on trucks
- In 1993 Denver built a new airport with a “state of the art” luggage system that replaced the human operators with robotic carts
  - The system failed
  - The airport could not open without a luggage system
  - The system was replaced (it took over a year)
  - The cost was almost $1B…. (yes one billion… 1994 dollars)
  - The company that designed the system went bankrupt

Exercise

- Fred sells bunches of flowers at the local shopping center.

- One day Fred's boss, Joe, tells Fred that at any time during the day he (Joe) will need to know:
  - how many bunches of flowers have been sold
  - what was the value of the most expensive bunch sold
  - what was the value of the least expensive bunch sold
  - what is the average value of bunches sold

Exercise


Selling flowers algorithm
START
Bunches = 0, Highest = 0
Lowest = 100,
Average = 0, Total = 0
Any Bunches to sell?
Yes
No
Sell bunch for price
increment bunch count
total = total + price
average = total / bunches
Is price > Highest?
Yes
No
Highest = price
Is price < Lowest?
Yes
No
Lowest = price
Page 1
STOP

Building Test Cases

3.6

Problem Solving: Test Cases

- To verify the correctness of a program, you must test it
  - A Test Case is a set of inputs that are used to verify whether the program output is correct in that specific case
- Testing is never 100% complete, but should cover all possible behaviors of the program
- Aim for complete coverage of all decision points (alternatives)
  - Create separate test cases for every programming alternative (e.g., domestic/international)
  - Create test cases for “boundary conditions”: the minimum (maximum) value, a value just above the minimum (max), just above the minimum (max), just above/below an program threshold, …
  - Create test cases for special values (0, 1, very large numbers, …)
  - Create test cases for invalid inputs (negative values, strings instead of numbers, empty values, …)

Make a Schedule…

- Make a reasonable estimate of the time it will take you to:
  - Design the algorithm
  - Develop test cases
  - Translate the algorithm to code and enter the code
  - Test and debug your program
- Leave some extra time for unanticipated problems
- As you gain more experience your estimates will become more accurate. It is better to have some extra time than to be late

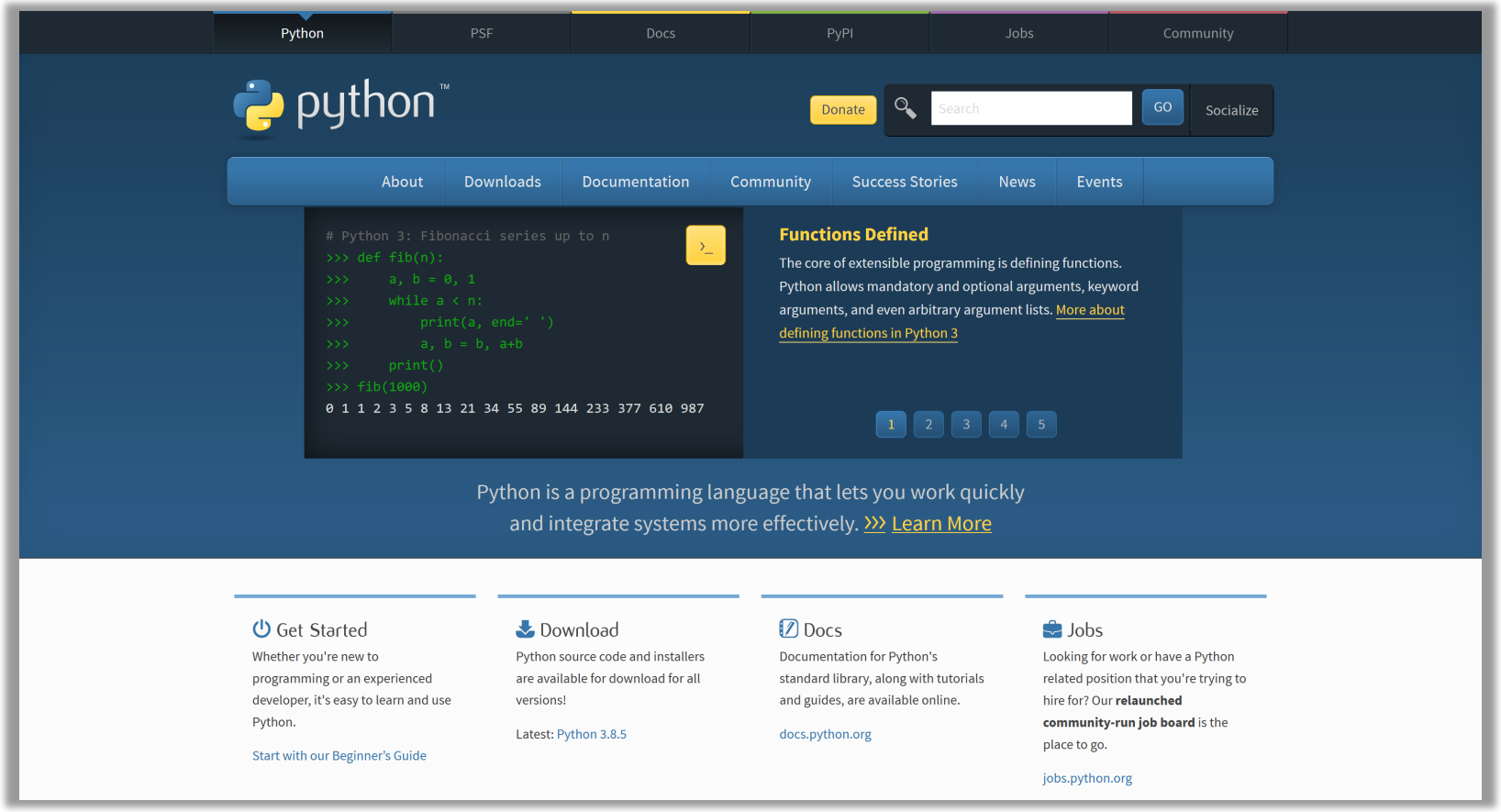
Introducing Pyton

1.3, 1.4,
1.5, 1.6

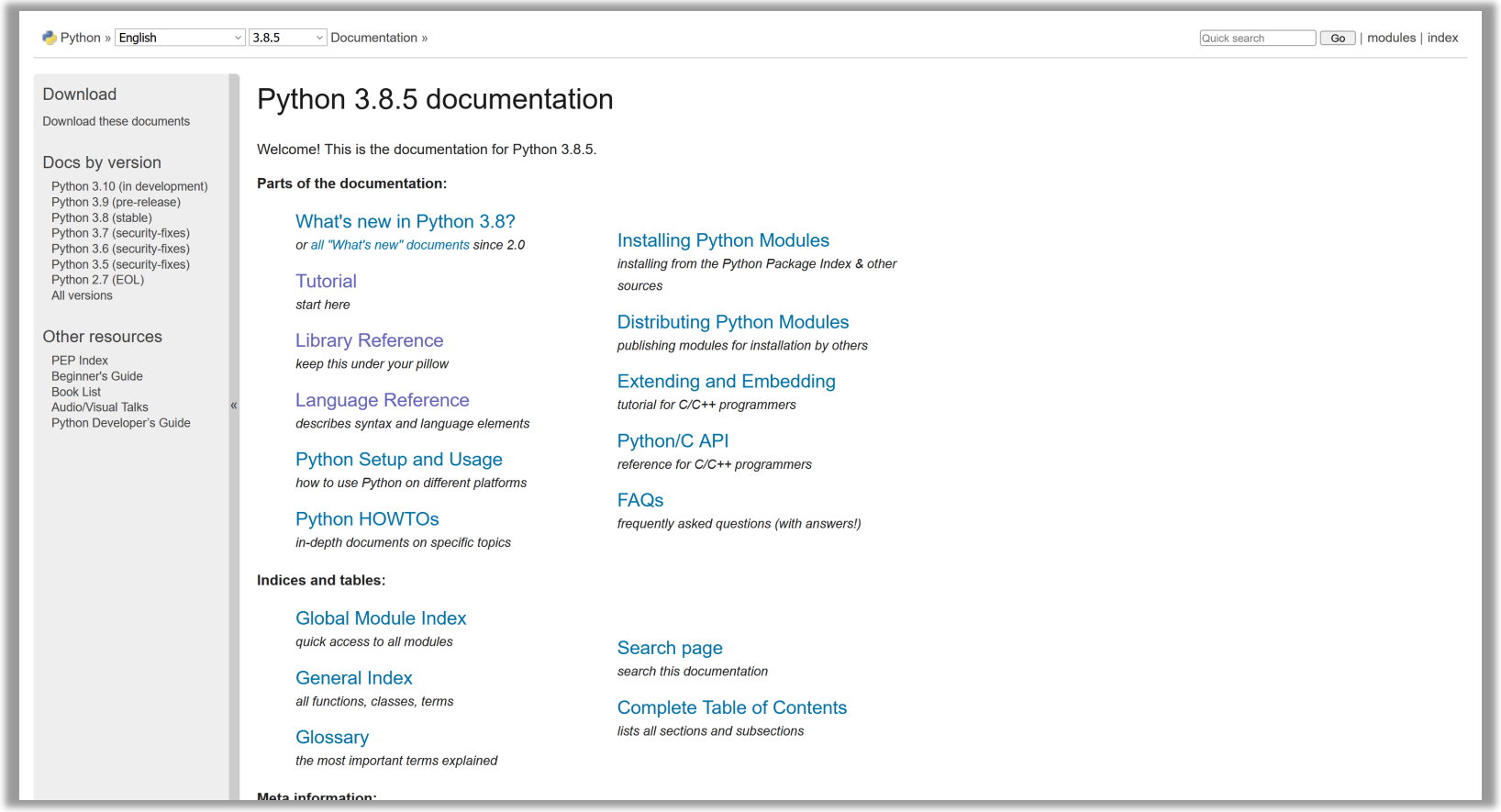
The Python Language

- In the early 1990's, Guido van Rossum designed what would become the Python programming language
- Van Rossum was dissatisfied with the languages available
  - They were optimized to write large programs that executed quickly
- He needed a language that could not only be used to create programs quickly but also make them easy to modify
  - It was designed to have a much simpler and cleaner syntax than other popular languages such as Java, C and C++ (making it easier to learn)
  - The Python environment featured an "all batteries included" approach, with the availability of many commonly used functions
  - Python is interpreted, making it easier to develop and test short programs
- Python programs are executed by the Python interpreter
  - The interpreter reads your program and executes it

https://www.python.org/



https://docs.python.org/



Programming Environments

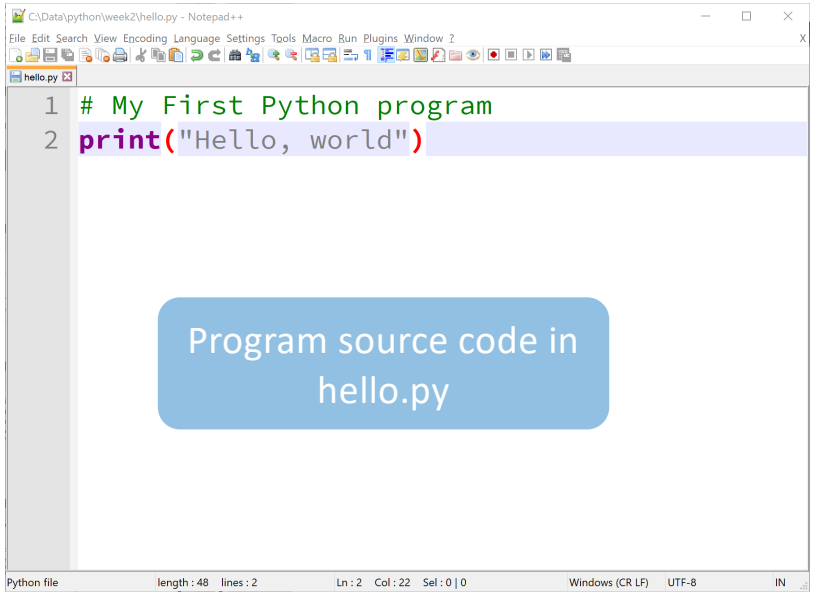
- There are several ways of creating a computer program
  - Using an Integrated Development Environment (IDE)
    - IDLE, PyCharm, Visual Studio Code, …
  - Using a text editor
    - Notepad, Notepad++, Atom, vi, gedit, …
- You should use the method you are most comfortable with
  - In this course we will use the PyCharm IDE or the repl.it on-line IDE
  - Book examples use the Wing IDE or the Spyder IDE

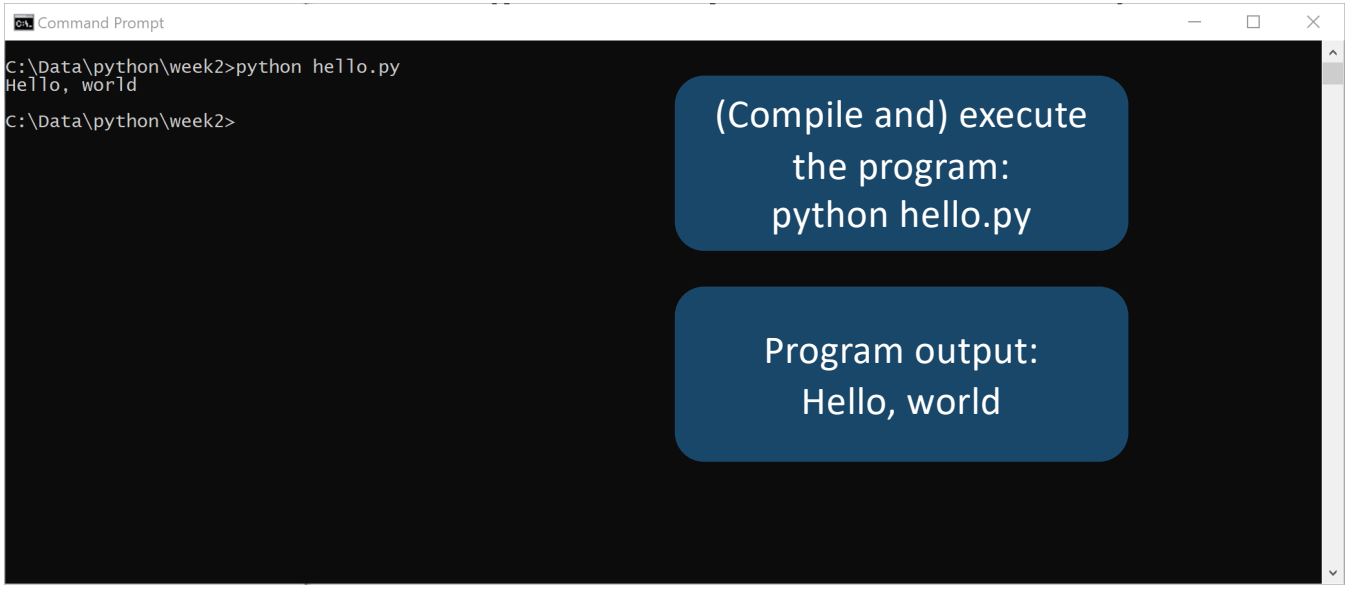
IDE components

- The source code editor can help programming by:
  - Listing line numbers of code
  - Syntax highlighting and coloring (comments, text…)
  - Auto-indent source code
  - Highlight syntax errors
  - Auto-completion
- Output window
  - The output generated by the program
- Debugger
  - Tools to help you search and correct logic errors in the program

Text editor programming

- You can use a simple text editor to write your source code
- Once saved as hello.py, you can use a console window to:
  - Compile & Run the program

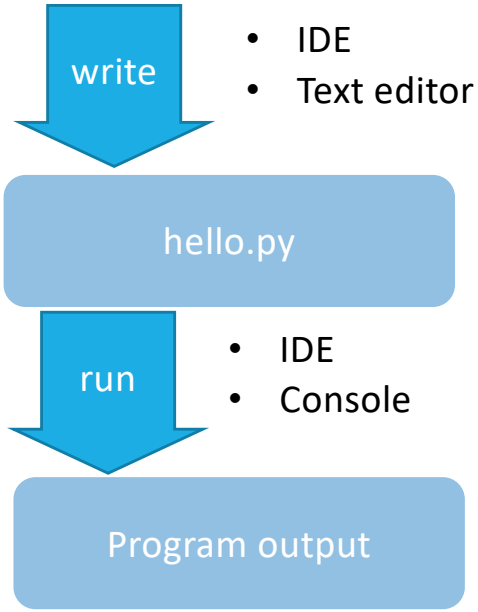




Your first program

- Traditional ‘Hello World’ program in Python
  - `print` is an example of a Python statement

```
1  # My first Python program.
2  print("Hello, World!")
3
```

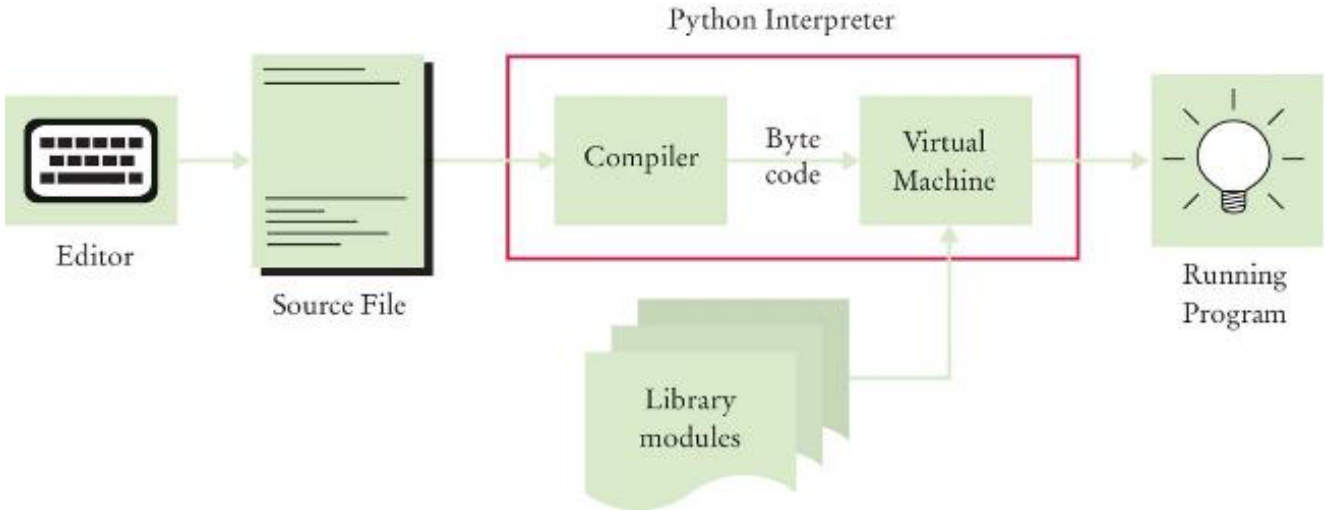


Writing/typing a Python program

- Careful about spelling - e.g., ‘`print`’ vs. ‘`primt`’
- PyTHon iS CaSe SeNsItiVe
- Spaces are important, especially at the beginning of a line (indentation)
- Lines beginning with # are comments (ignored by Python)

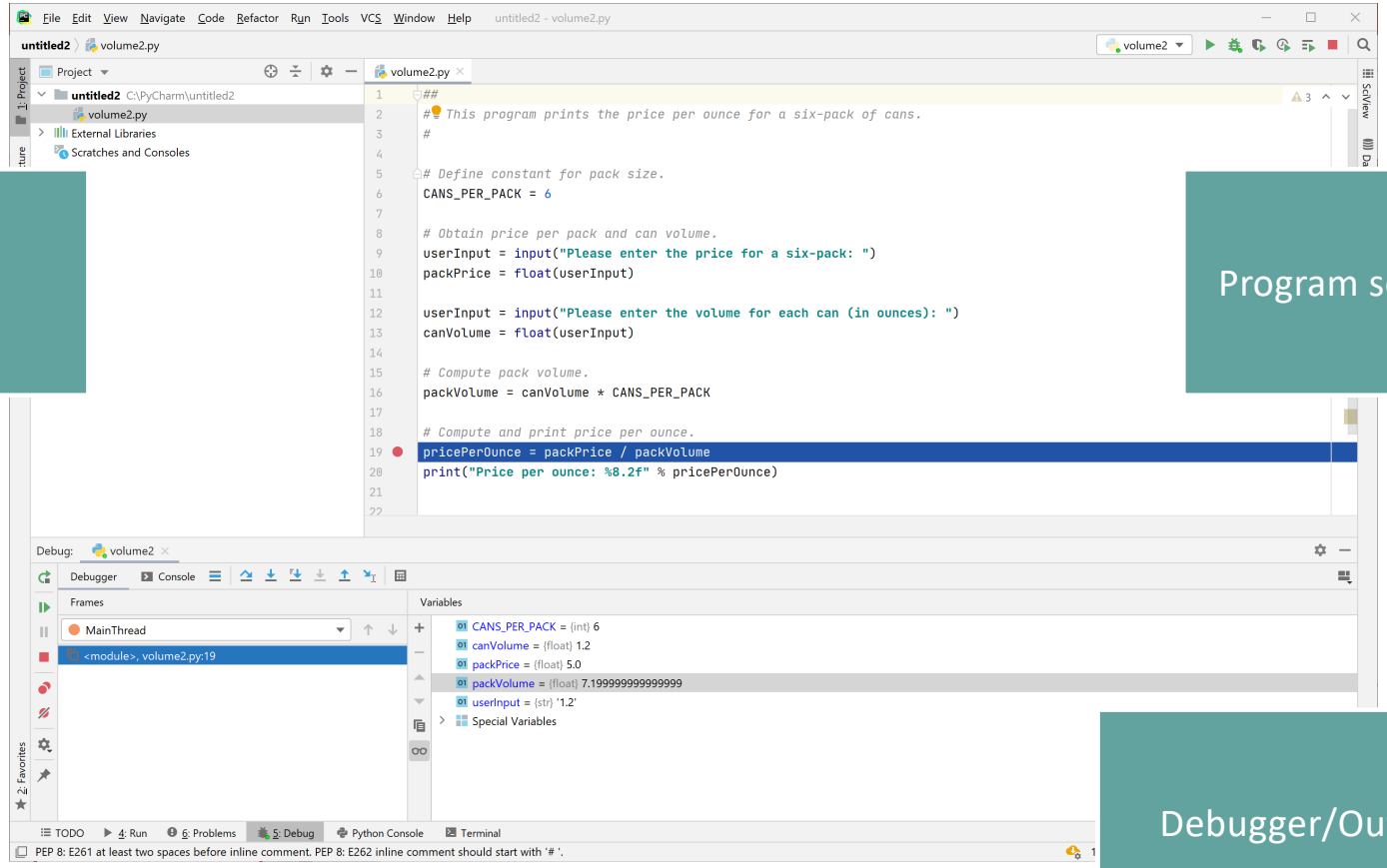
Source Code to a Running Program

- The compiler reads your program and generates byte code instructions (simple instructions for the Python Virtual machine)
  - The Python Virtual machine is a program that is similar to the CPU of your computer
  - Any necessary libraries (e.g. for drawing graphics) are automatically located and included by the virtual machine



The PyCharm IDE

Project files



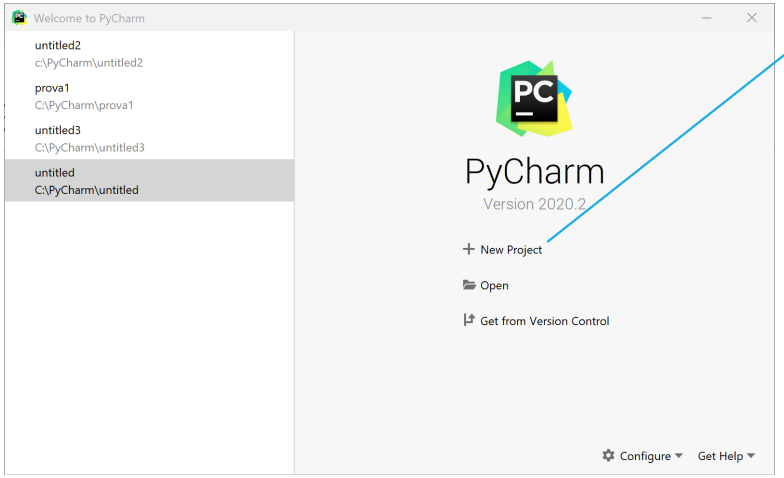
Program source code

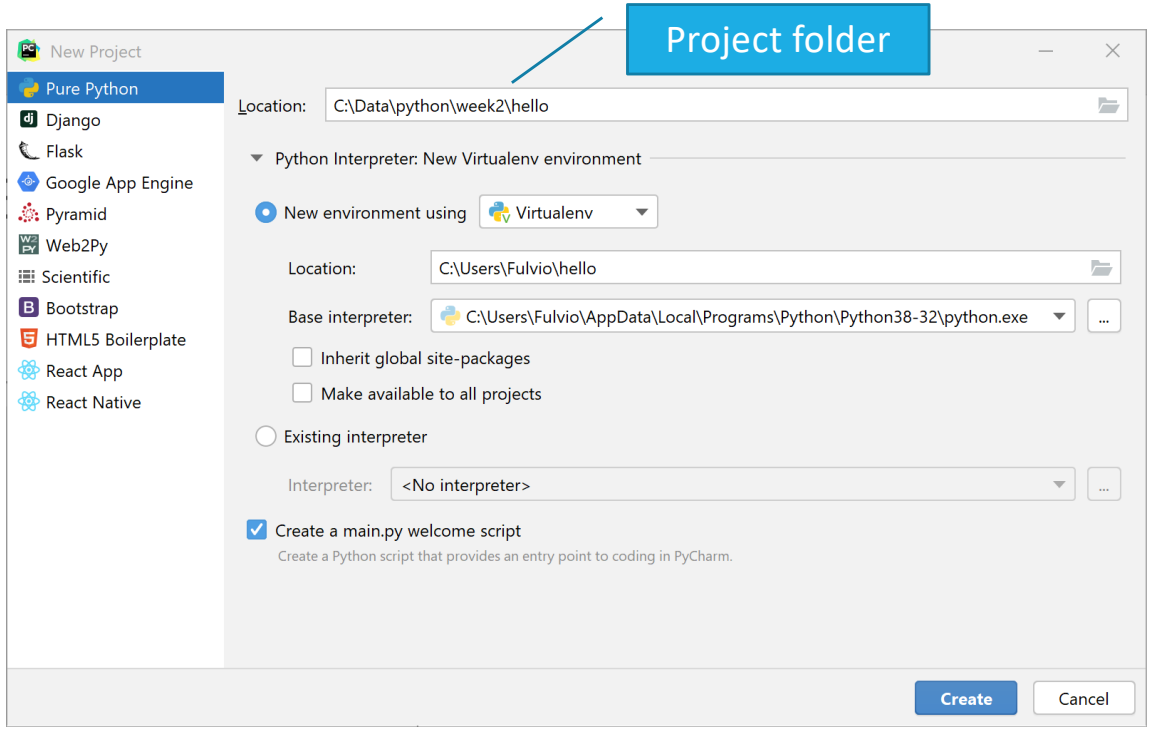
Debugger/Output

Project vs Program vs File

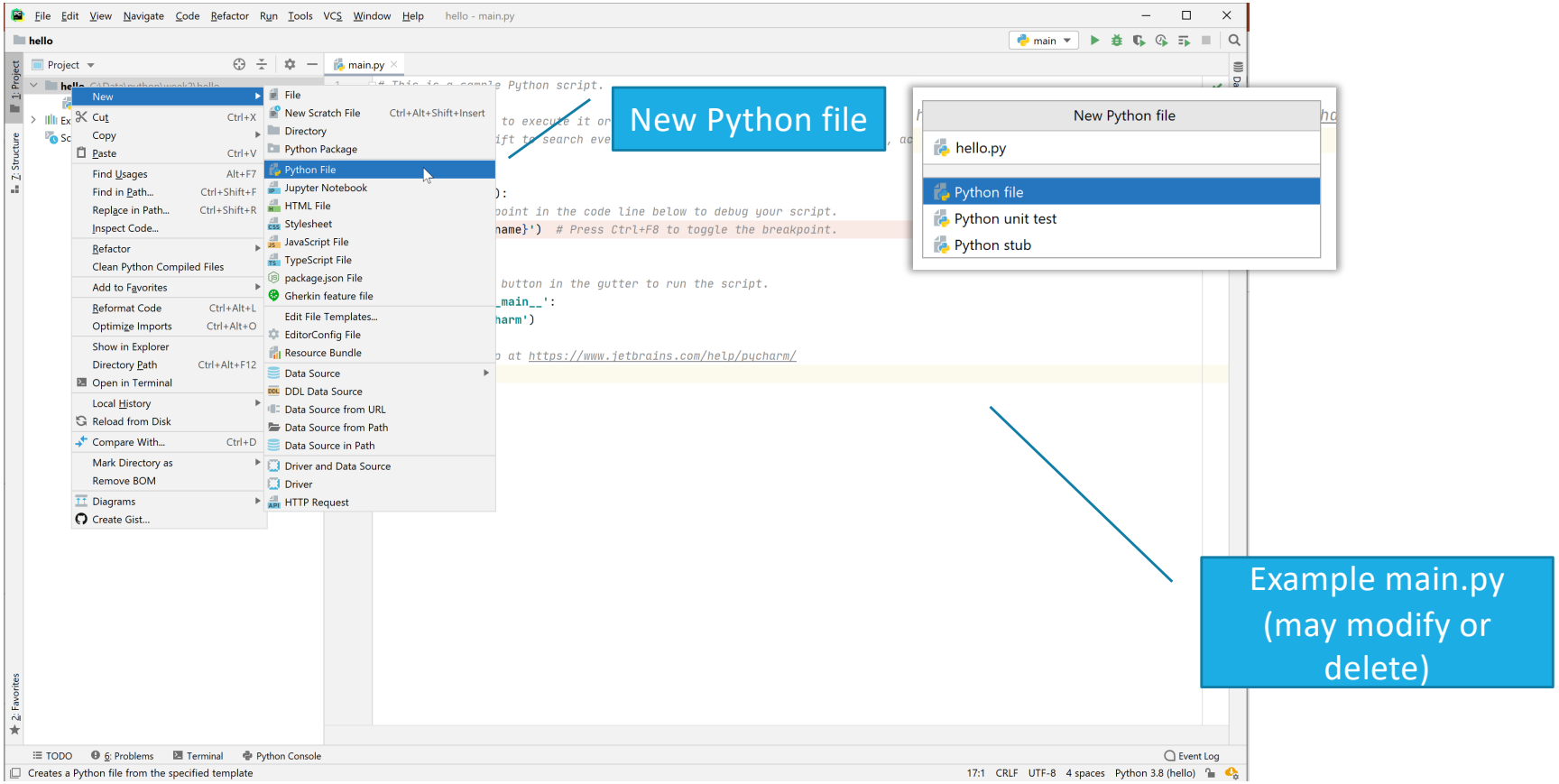
- A single Program may be very large, and will be composed of may different files
- IDEs allow us to group a set of related files into a “Project”
- Every time we want to create a new program, we must
  - Create a new Project
  - Create one (or more) Python Files inside the Project

Programming with the PyCharm IDE

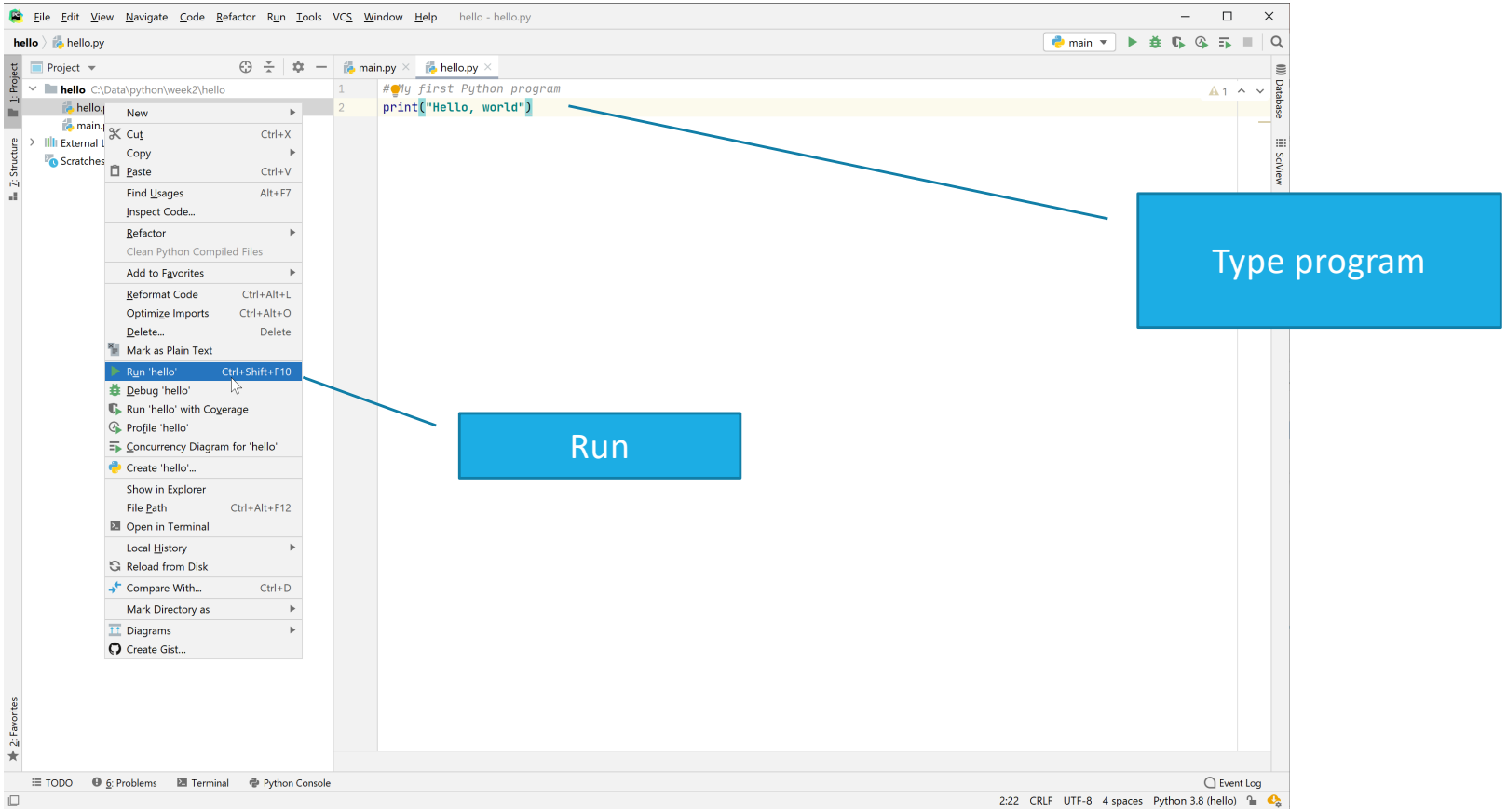




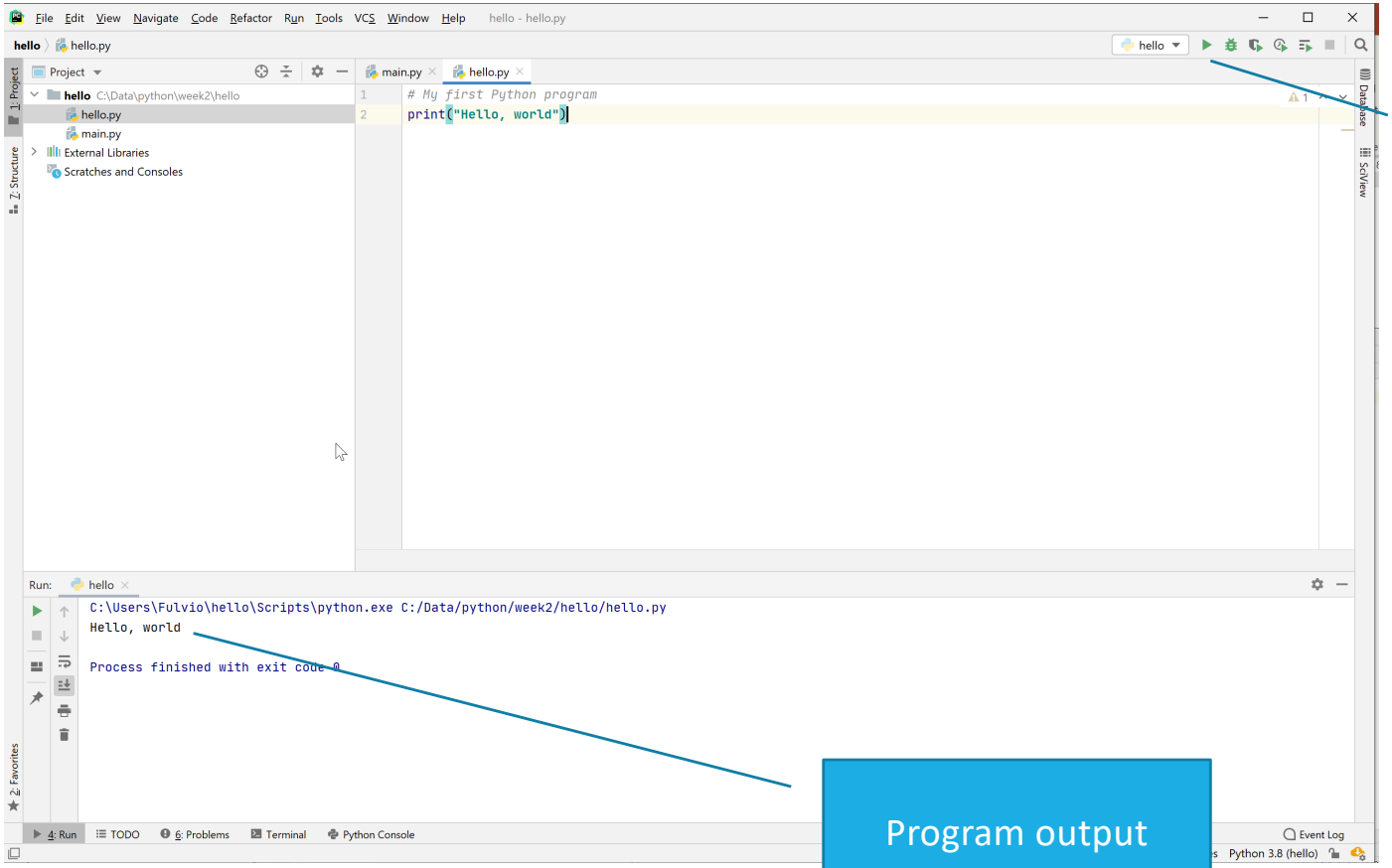
Programming with the PyCharm IDE



Programming with the PyCharm IDE



Programming with the PyCharm IDE



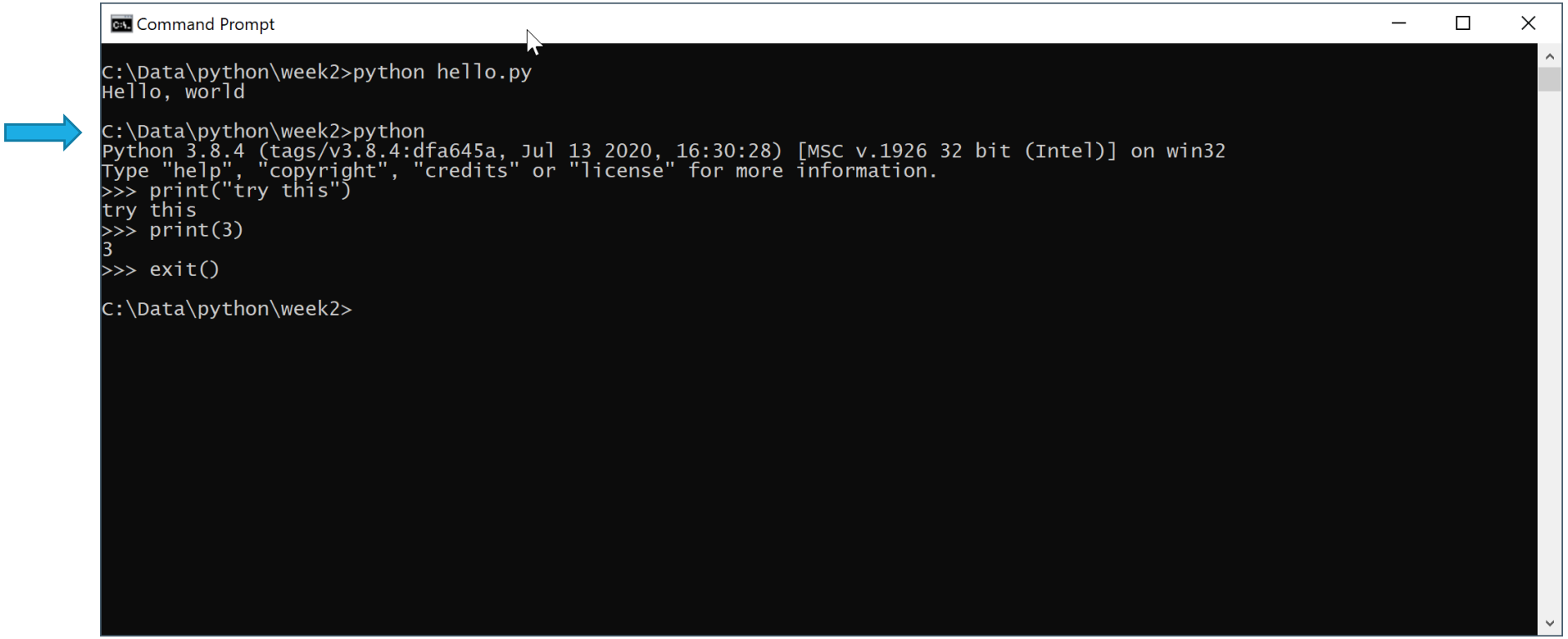
Organize your work

- Your ‘source code’ is stored in .py files
- Create a folder for this course
- Create one **project folder** per program inside the course folder
  - A program can consist of several .py files
- Backup your files regularly
  - To a USB flash drive (or more than one)
  - To a network drive (cloud service) or external hard drive
  - Do it. Really.

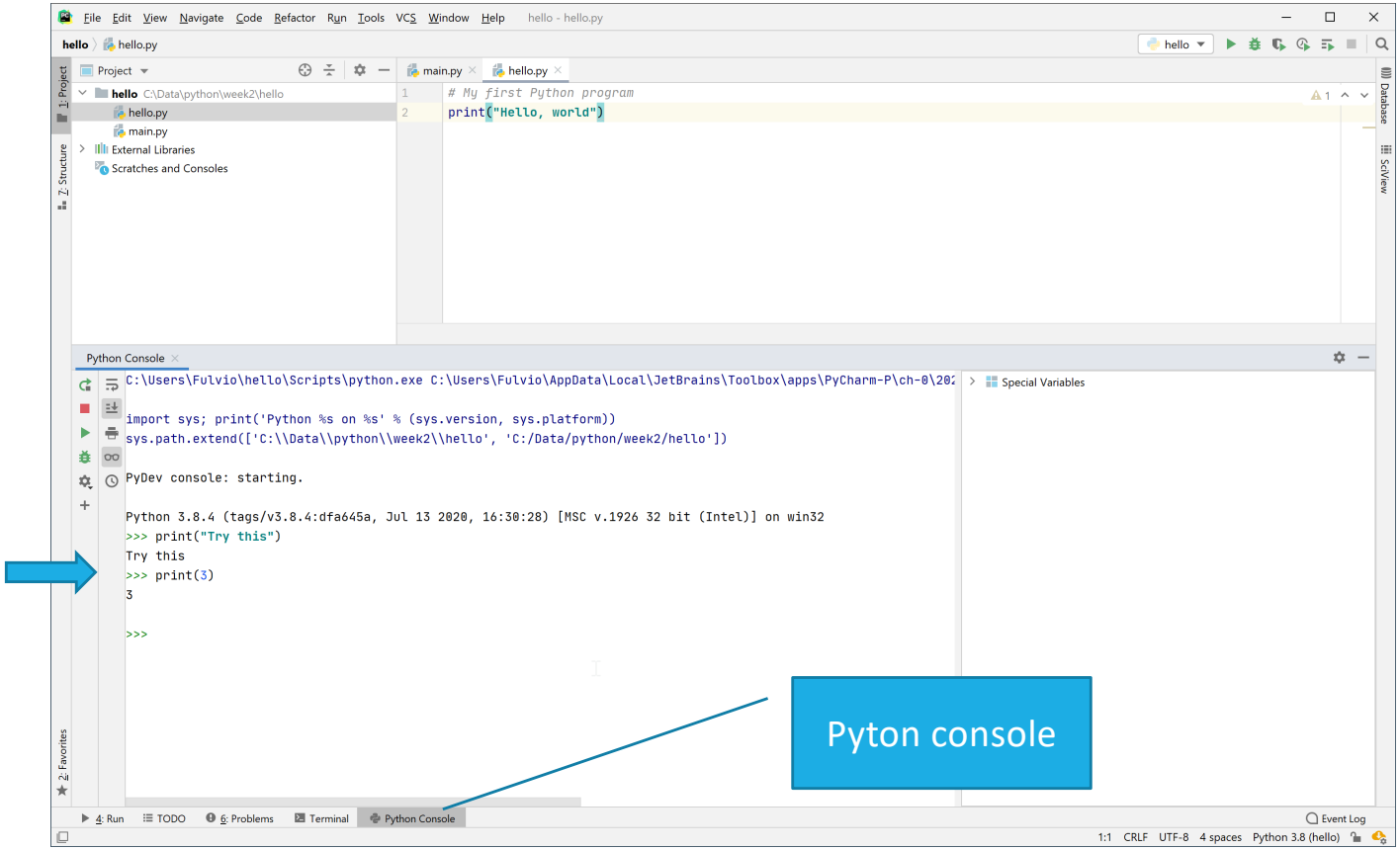
Python interactive mode

- The Python interpreter loads a file and executes all the instructions at once
  - Similar to other (compiled) languages
- Alternatively: in **interactive mode**, you can run instructions one at a time
  - It allows quick ‘test programs’ to be written
  - Try and experiment instructions
  - Interactive mode allows you to write python statements directly in the console window

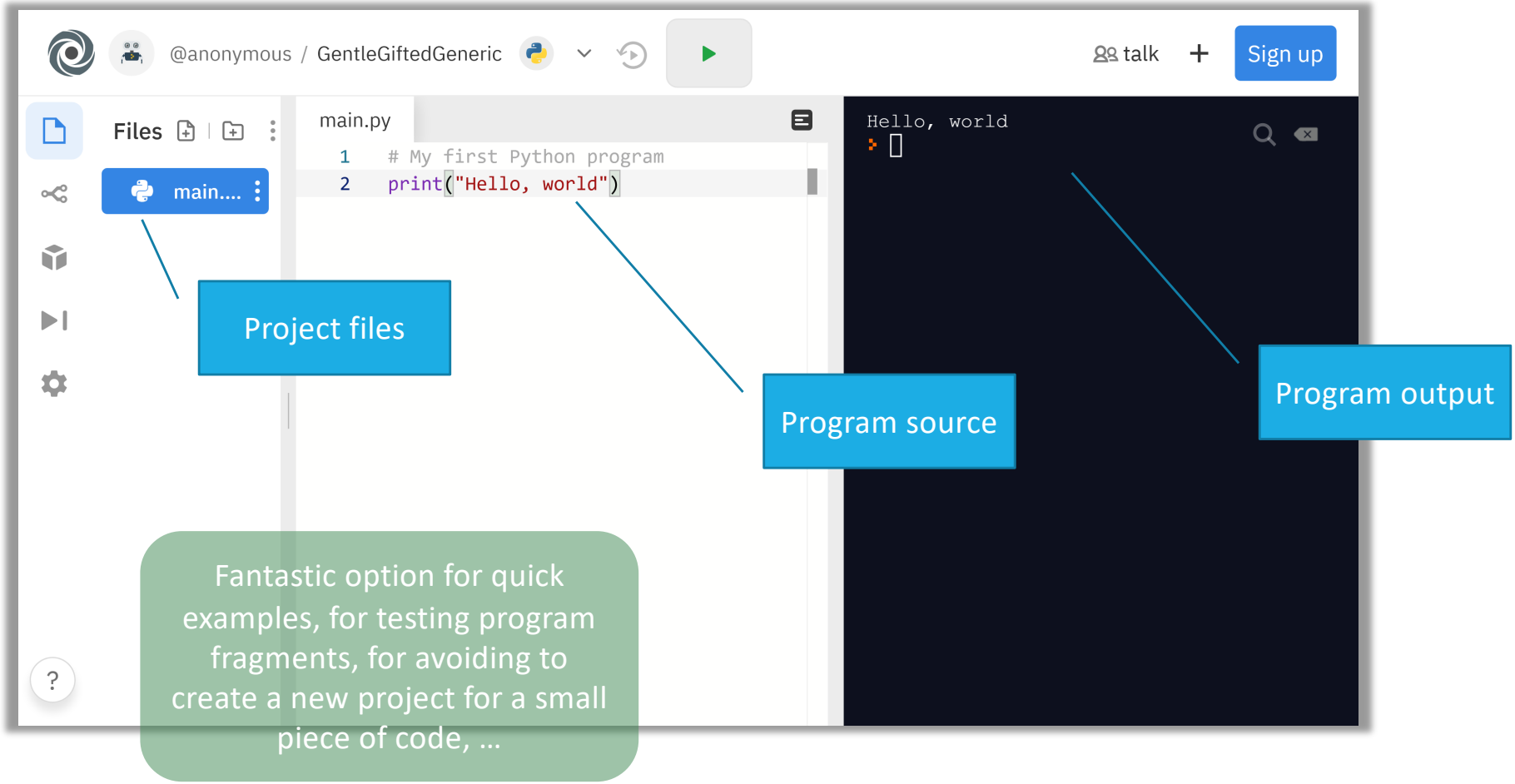
Python interactive mode



Python interactive mode

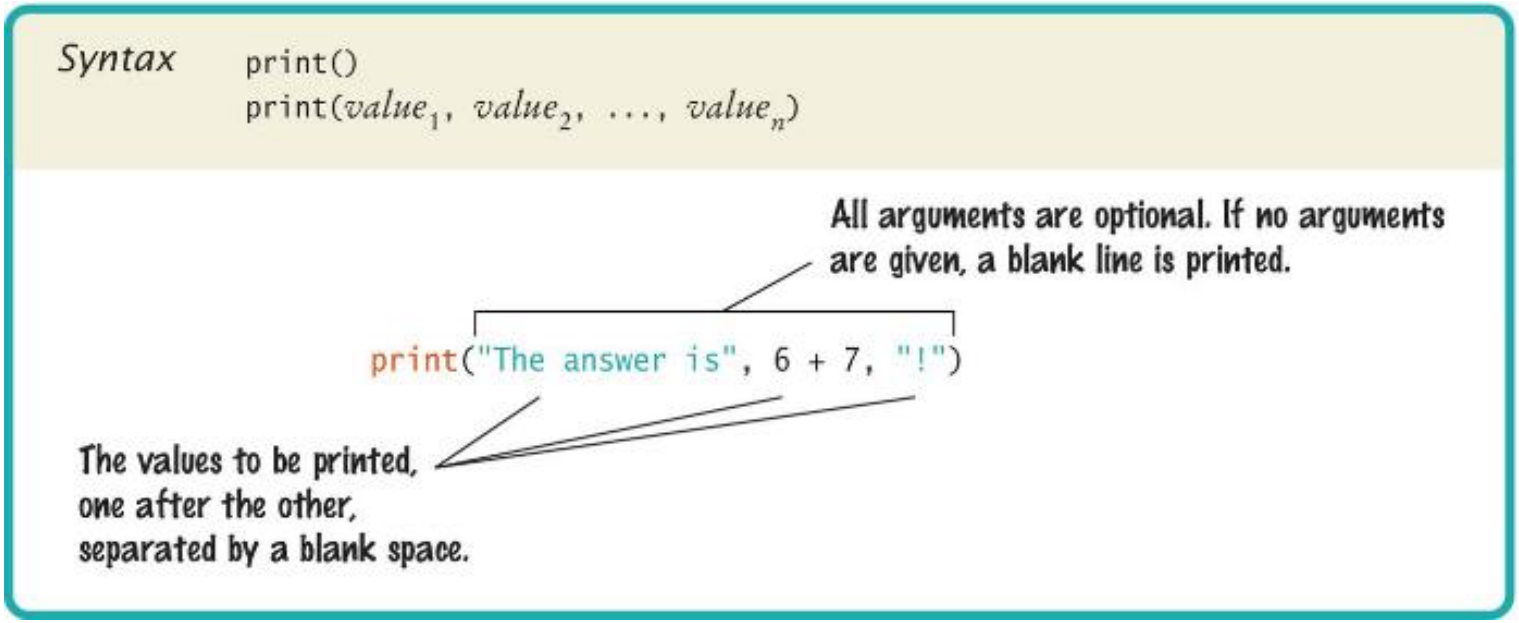


On-line IDE: https://repl.it/



Basic Python Syntax: Print

- Using the Python `print()` function
  - A function is a collection of programming instructions (with a **name**) that carry out a particular task (in this case to print a value onscreen)
  - It's code that somebody else wrote for you!



Syntax for Python Functions

- To use, or call, a function in Python you need to specify:
  - The name of the function that you want to use
    - In the previous example, the name was `print`
  - All values (arguments) needed by the function to carry out its task
    - in this case, `"Hello World!"`
  - Arguments are enclosed in parentheses
  - Multiple arguments are separated with commas.

Strings

- A *sequence of characters* enclosed in quotations marks are called a *string*
  - May be enclosed in `'single quotes'`
  - May be enclosed in `"double quotes"`

More examples of the `print` function

- Printing numerical values
  - `print(3 + 4)`
  - Evaluates the expression 3 + 4 and displays 7
- Passing multiple values to the function
  - `print("The answer is", 6 * 7)`
  - Displays `The answer is 42`
  - Each value passed to the function is displayed, one after another, with a blank space after each value
- By default the print function starts a new line after its arguments are printed
  - `print("Hello")`
  - `print("World!")`
  - Prints two lines of text:
    - `Hello`
    - `World!`

Our Second Program (printtest.py)

```
##
#  Sample Program that demonstrates the print function
#
#  Prints 7

print(3 + 4)

#  Print Hello World! on two lines
print("Hello")
print("World!")

#  Print multiple values with a single print function call
print("My favorite numbers are", 3 + 4, "and", 3 + 10)

#  Print Hello World! on two lines
print("Goodbye")
print()
print("Hope to see you again")
```

Errors

COMPILE-TIME ERRORS

OR **SYNTAX** ERRORS

- Spelling, capitalization, punctuation
- Ordering of statements, matching of parenthesis, quotes, indentation, …
- No executable program is created by the compiler
- Correct first error listed, then compile again
  - Repeat until all errors are fixed
- Usually detected and highlighted by the IDE

RUN-TIME ERRORS

OR **LOGIC** ERRORS

- The program runs, but produces unintended results
- The program may ‘crash’
- The most difficult to find and correct
  - Even for experienced programmers

Syntax Errors

- Syntax error are caught by the compiler
- What happens if you
  - Miss-capitalize a word `Print("Hello World!")`
  - Leave out quotes `print(Hello World!)`
  - Mismatch quotes `print("Hello World!')`
  - Don’t match brackets `print('Hello'`
- Type each example above in the IDE
  - In the program source code
  - In the interactive Python console
  - What error messages are generated?

Logic Errors

- What happens if you
  - Divide by zero `print(1/0)`
  - Misspell output `print("Hello, Word!")`
  - Forget to output Remove line 2
- Programs will compile and run
  - The output may not be as expected
- Type each example above in the IDE
  - What error messages are generated?