COMPSCI 111 / 111G

Mastering Cyberspace:
An Introduction to Practical Computing

Introduction to Programming and Python

Routine Activities

We spend much of our time engaging in routine activities:

- Morning activities like taking a shower and getting dressed
- Getting to the university or work and returning home
- Organizing and holding a party for our friends
- Driving or flying to visit our parents or siblings
- Having lunch or dinner at a restaurant or fast food outlet

We may value novelty, but familiar routines make our life much simpler than it would be without them.

Familiar Automated Routines

We rely on devices for many routine activities, including:

- Alarm clocks or cell phones to wake up in the morning
- Microwave ovens to cook or reheat our meals
- Tivo boxes to record our favorite TV programs
- Thermostats to control air conditioners and heaters
- Washers and dryers to clean and dry our clothes
- iPods to play the songs or lectures we want to hear

These artifacts *store* and *execute* routines so we do not have to carry them out ourselves.

Stored Programs

How do our digital devices know how to carry out such routine activities?

- Basic idea: *We can store the steps in a routine activity in memory in ways that a computer can execute.*

This notion of a *stored program* is one of the basic insights that underlies computing and information technology.

This concept is also central to all computational theories of human thinking and intelligence.

Storing and Interpreting Programs

We need two things to allow routine activity on a computer:

- Encode and store the program in memory:
  - Giving the program a name or index so we can access it;
  - Specifying the program's initial, intermediate, final steps.
- Access and run the program on demand:
  - Retrieving the program by its name or index;
  - Passing any arguments to the program;
  - Executing each of the program's steps in turn; and
  - Halting the program and returning any results.

These ideas apply not only to routine behavior on silicon devices, but also to our own everyday activities.

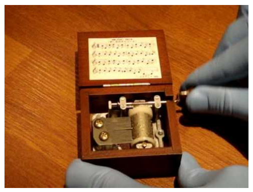
Recipe for Jamaican Curry Chicken

Ingredients: 1/2 chicken, 4 stems thyme, 1 onion, 5 tbsp. curry, 2 cloves garlic, 3 tbsp. seasoning, 3 tbsp. black pepper, 1 tbsp. chicken seasoning, 3 cubed potatoes,1 lemon,1 tbsp. vegetable oil, 2 tbsp. butter, 2 c. water, 1/2 Scotch bonnet pepper

1. Wash chicken with lemon and cut into bite-sized pieces.
2. Season with all dry ingredients.
3. Chop all herbs and add to chicken (use hands to rub in seasonings, and let sit in refrigerator for 1/2 hr.).
4. Place chicken, water, and oil in a pot, stir, cook on high until it comes to a boil, stir, lower heat until chicken is almost cooked.
5. Add potatoes and butter.
6. Cook until water is reduced and potatoes are tender.
7. Serve over steamed white rice.

Music Boxes as Stored Programs

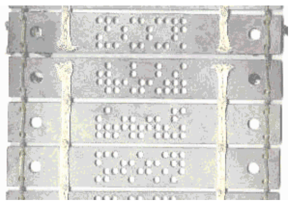
Music boxes use a revolving cylinder with adjustable pins that hit the tuned teeth of a steel comb.



The adjustable pins are a stored program. This idea goes back at least to Nicholas Vallin's 1598 musical wall clock.

The Jacquard Loom

Around 1800, Joseph-Marie Jacquard first introduced punched cards to control a loom for weaving silk cloth.



Each card specified one row in the design; a linked chain of such cards was a stored program for weaving a pattern.

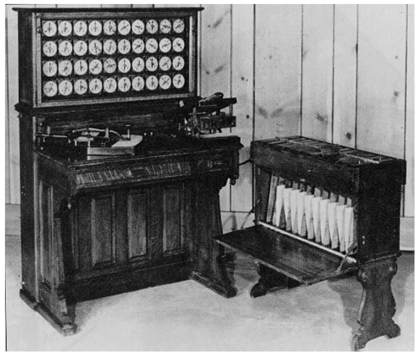
Stored Programs in Player Pianos

In 1863, Forneaux reported the first practical self-playing piano, which used a perforated cardboard book to control the keys.

Later player pianos used paper rolls to describe a song's notes; these replaceable rolls were stored programs.

Hollerith's Tabulator

Herman Hollerith developed a *mechanical tabulator*, driven by punched cards, to process the 1890 US Census.



Unlike earlier stored programs, these supported not some routine physical activity but rather routine calculation.

History of Stored Programs

Stored programs have a diverse and illustrious history:

- Joseph-Marie Jacquard (~1800) introduces punched cards to control a loom for weaving silk cloth.
- Herman Hollerith (~1890) uses punched cards for tabulating data from the US Census.
- Alan Turing (1937) introduces the idea of a machine that can store and interpret a program as *data* in memory.
- William Mauchly and J. P. Eckert (~1946) design the first stored-program *digital* computer.
- First commercial programmable computer (UNIVAC 1) goes on the market (1951).

Computer programming builds on these varied innovations.

Media for Stored Programs

Note that the media used for stored programs has evolved over time:

- Metal cylinders for music players
- Wooden cards for looms and paper rolls for pianos
- Punched paper cards for tabulating machines
- Magnetic drums for early digital computers
- Semiconductor memory for later digital computers

Note that we can store the *same program* – instructions for the same routine behavior – on very different media.

This suggests that programs exist at a level of abstraction that is *independent* of their physical encoding.

Programming Languages

Any stored program must be stated in some language.

Since computers cannot yet understand natural languages, we require something more formal and less ambiguous.

A *programming language* is a formal notation that lets us tell computers how to carry out routine activities.

There are many programming languages, some of them in existence since the 1950s:

- Fortran, Algol, Basic, Visual Basic
- C, C++, C#, Java
- Lisp, Prolog, OPS … and many, many others

This course will focus on *Python*, a fairly recent language.

Programs as Ordered Statements

A stored program for carrying out a routine activity has:

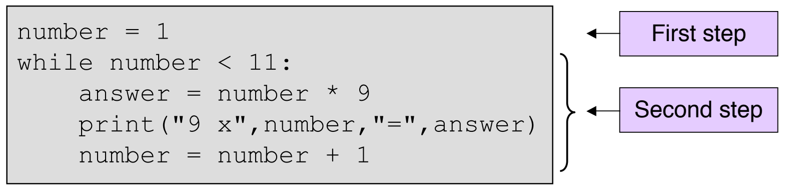
- A *name* by which one can access the program.
- A set of inputs or *arguments* on which it operates.
- *Initialization steps* that get the program ready.
- A list of *operations* that must occur in sequence.
- *Halting steps* that end the program and return results.

We often refer to each step in a program as a *statement*.

Thus, the body of a program consists of a sequence of statements, much as in a recipe.

Ordered Statements: Python Example

Here is a simple Python program that has ordered steps.



```
number = 1
while number < 11:
    answer = number * 9
    print("9 x",number,"=",answer)
    number = number + 1
```

You can ignore the details for now; we will talk about what each step means later, in another lecture.

The important point is that you can specify steps in the order you want Python to carry them out.

Building Blocks of Programs

Any programming language must make a commitment to three building blocks that make up programs:

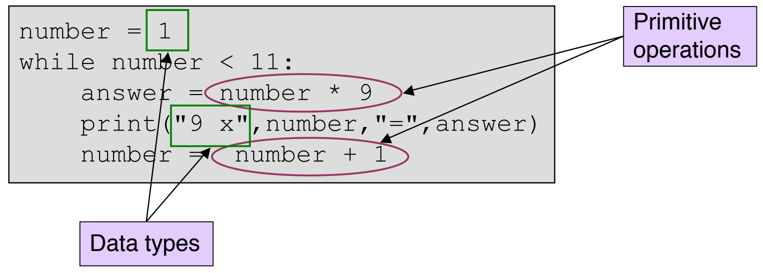
- *Types of data* over which programs operate;
- *Primitive operations* that inspect / manipulate those data;
- A *syntax* for *combining* these operations into programs.

Different languages make different commitments about each of these design decisions.

We will consider Python's position on each dimension.

Building Blocks: Python Example

Our example Python program also illustrates data types, primitive operations, and syntax for combining them.



```
number = 1
while number < 11:
    answer = number * 9
    print("9 x",number,"=",answer)
    number = number + 1
```

Python uses reserved terms like ‘while’ and indentation to specify how to combine operations and data.

Data Types in Python

Python programs operate on data that comes in different forms:

- Integers
  - Numbers without a decimal point
  - E.g., –100, 0, 45
- Floating-point numbers
  - Numbers with a decimal point
  - E.g., –1.00002, 0.0, 4.5, 45.0
- Strings
  - Sequence of characters in text (ASCII or Unicode)
  - Enclosed in quotation marks
  - E.g., "Hello", "Goodbye", "the dog barked"
- Lists and list structures
  - Ordered sets of elements
  - E.g., ["the", "dog", "barked"], [[1, 2], [3, [4, 5]]]

Arithmetic Operators in Python

Python includes a number of built-in functions for making numeric calculations:

- Adding (`+`) and multiplying (`*`) two numbers:
  - `5 + 3 => 8 , 5 * 3 => 15`
- Subtracting (`-`) and dividing (`/`) two numbers:
  - `5 - 3 => 2 , 5 / 3 => 1.6666666666666667`
- Integer division (`//`) and remainder (`%`) for two numbers:
  - `5 // 2 => 2 , 5 % 2 => 1`
- Exponentiation (`**`) of one number by another:
  - `5 ** 3 => 125`

One can combine such expressions using standard rules of precedence [e.g., `1 + 2 * 3 => 7 , (1 + 2) * 3 => 9`].

String/List Operators in Python

Python also includes built-in functions for operating over strings and lists, such as those for:

- Concatenating two strings (+)
  - **`"The" + " " + "dog" => "The dog"`**
- Repeating a given string (`*`) multiple times
  - **`"Hear Ye " * 3 => "Hear Ye Hear Ye Hear Ye "`**
- Concatenating two lists (+)
  - **`[1, 2, 3] + [4, 5] => [1, 2, 3, 4, 5]`**
- Repeating a given list (`*`) multiple times
  - **`E.g., ["a", "b"] * 2 => ["a", "b", "a", "b"]`**

Other operators are available for manipulating, inspecting, and extracting content from data of both types.

Compilers and Interpreters

Early programming languages were low level and easy for digital machines to interpret, but difficult for humans.

More recent languages have been high level and better for humans, but must be translated into lower-level ones.

Computer science has explored two general approaches:

– Program *compilers* that translate an entire file at once into a language that the computer CPU can process.

– Program *interpreters* that translate one statement at a time, running the code as it is converted.

Compilation often produces more efficient programs, but interpreters support more interaction and debugging.

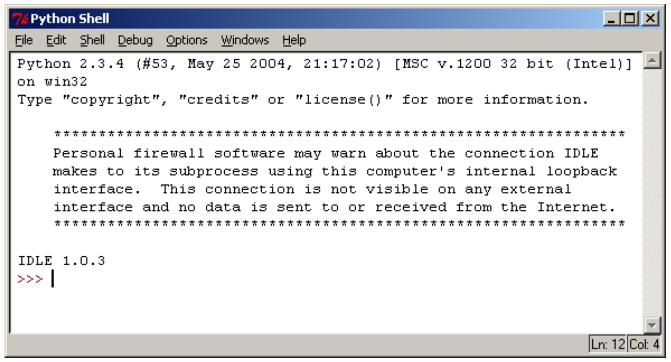
Python is an *interpretive* programming language.

Interactive Execution in Python

IDLE is an *integrated development environment* for Python that incorporates:

– A text editor for entering statements

– An interpreter to execute these statements



```
Python 2.3.4 (#53, May 25 2004, 21:17:02) [MSC v.1200 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.

    ****************************************************************
    Personal firewall software may warn about the connection IDLE
    makes to its subprocess using this computer's internal loopback
    interface.  This connection is not visible on any external
    interface and no data is sent to or received from the Internet.
    ****************************************************************

IDLE 1.0.3
>>> 
```

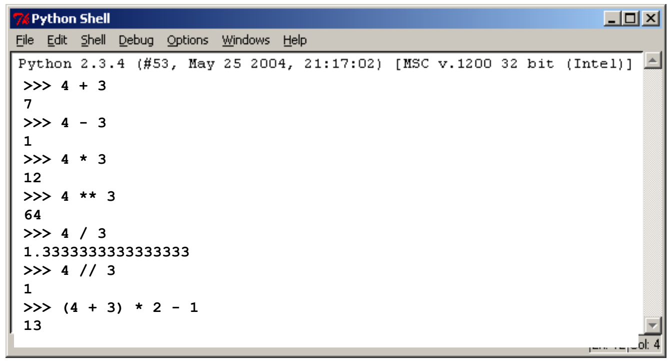
Running Python Code

Python offers users two ways to create and run programs:

- *Interactive execution*
  - Type a statement directly at the prompt
  - Python executes statement when you hit <Enter>
  - Useful for experimentation
  - Good for learning the language
- *Defining and using stored programs*
  - Type a sequence of statements into a file
  - Save the file with the file extension .py
  - Load the file using *import* function or *run* in IDLE
  - Call the program under the desired conditions

Interactive Execution in Python

You can enter a statement directly into IDLE and Python will return the value it produces.

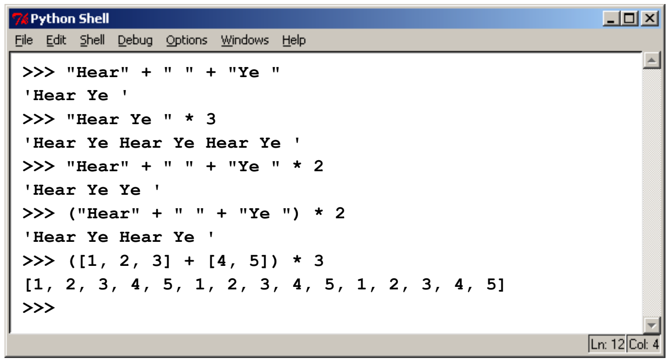


```
Python 2.3.4 (#53, May 25 2004, 21:17:02) [MSC v.1200 32 bit (Intel)]
 >>> 4 + 3
 7
 >>> 4 - 3
 1
 >>> 4 * 3
 12
 >>> 4 ** 3
 64
 >>> 4 / 3
 1.3333333333333333
 >>> 4 // 3
 1
 >>> (4 + 3) * 2 - 1
 13
```

Here are some examples of primitive arithmetic operations, along with one instance of combining them.

Interactive Execution in Python

You can also enter statements into IDLE that use string and list operators, and Python will return their values.



```
>>> "Hear" + " " + "Ye "
'Hear Ye '
>>> "Hear Ye " * 3
'Hear Ye Hear Ye Hear Ye '
>>> "Hear" + " " + "Ye " * 2
'Hear Ye Ye '
>>> ("Hear" + " " + "Ye ") * 2
'Hear Ye Hear Ye '
>>> ([1, 2, 3] + [4, 5]) * 3
[1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5]
>>>
```

Here are examples of string and list operations, including some complex statements.

Review of Key Ideas

The idea of a stored program is a powerful innovation that:

- Underlies all of modern computing and information technology
  - But has a long history that goes back to at least 1598
- Lets us automate complex but routine activities on computers
  - But also accounts for routine human behavior
- Can be implemented in many programming languages
  - But one need not master them to appreciate the key ideas

We rely increasingly on programs stored in digital devices, but we have always used them to carry out routine tasks.

Review of Key Ideas

A programming language lets us specify routine activity by:

- Specifying the *name* and *arguments* of a program
- *Initialization*, *sequential*, and *halting* steps for the activity
- Describe steps as *combinations* of primitive *operations* on alternative *data types* using an unambiguous *syntax*
- Run the stored program by translating its steps into lower-level instructions using a compiler or interpreter

Python is a recent but popular interpretive language with a simple syntax that is easy to use.

You will learn how to create, store, load, and run Python programs in your laboratory exercises.