ICQB

Introduction to Computational & Quantitative Biology (G4120)
Fall 2022

Oliver Jovanovic, Ph.D.
Columbia University
Department of Microbiology & Immunology

Lecture 4: Introduction to Programming
October 4, 2022

History of Programming

1843 Lady Ada Lovelace writes one of the first computer programs for Charles Babbage's Analytical Engine.

1936 Alan Turing develops the theoretical concept of the Turing Machine, forming the basis of modern computer programming.

1943 Plankalkül, the first formal computer language, is developed by Konrad Zuse, a German engineer, which he later applies to, among other things, chess.

1945 John von Neumann develops the theoretical concepts of shared program technique and conditional control transfer.

1949 Short Code, the first computer language actually used on an electronic computer, appears.

1951 A-O, the first widely used complier, is designed by Grace Hopper at Remington Rand.

1954 FORTRAN (FORmula TRANslating system) language is developed by John Backus at IBM for scientific computing.

1958 ALGOL, the first programming language with a formal grammar, is developed by John Backus for scientific applications

1958 LISP (LISt Processing) language is created by John McCarthy of MIT for Artificial Intelligence (AI) research.

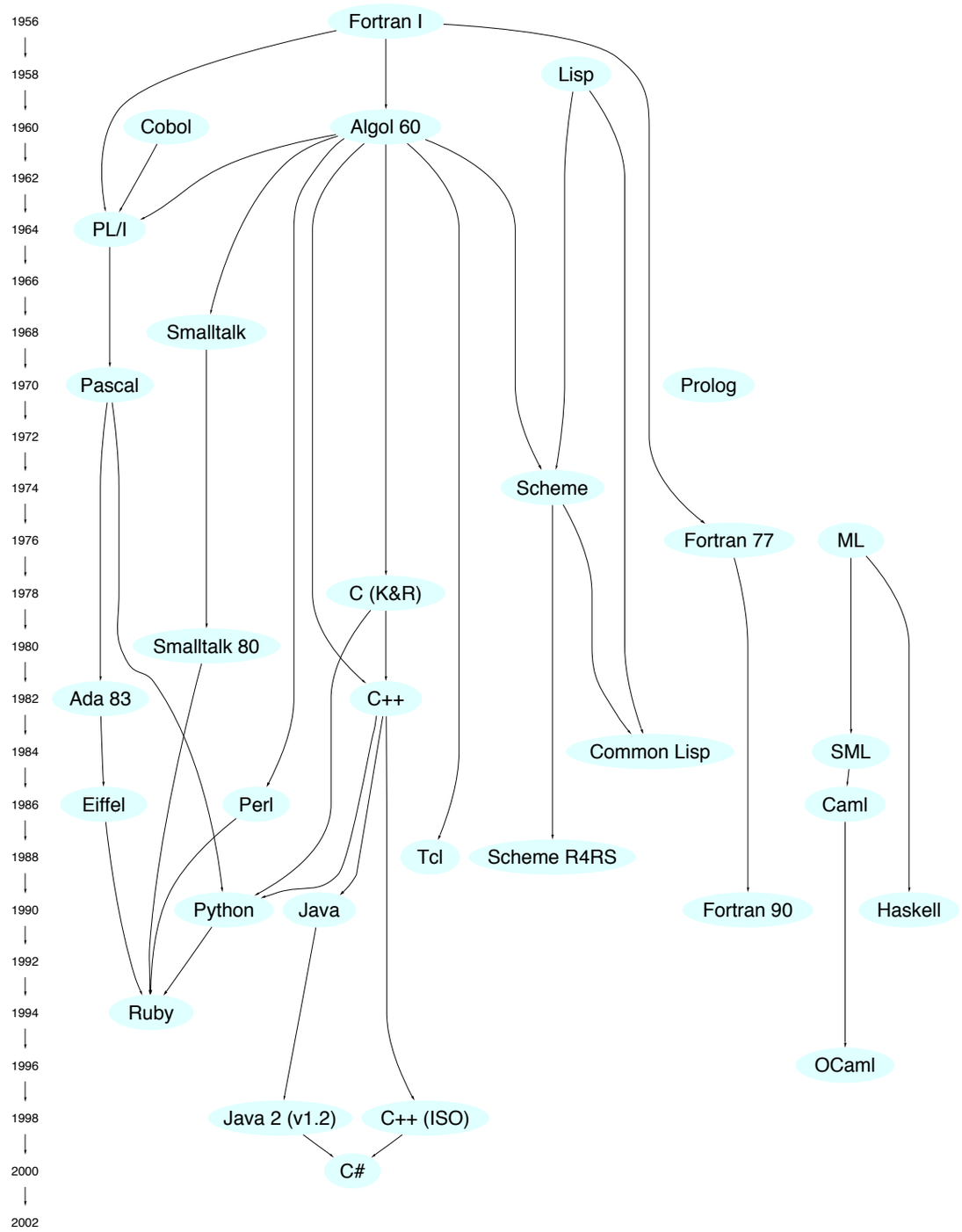
1959 COBOL is created by the Conference on Data Systems and Languages (CODASYL) for business programming, and becomes widely used with the support of Admiral Grace Hopper.

1964 BASIC (Beginner's All-purpose Symbolic Instruction Code) is created by John Kemeny and Thomas Kurtz as an introductory programming language.

1965 Structured programming is defined by Edsger Dijkstra.

1968 Pascal is created by Niklaus Wirth as a teaching language.

Evolution of Programming Languages



Modern Programming Languages

1972 Prolog (Programming Logic) is developed as result of logic theorem research. It has become the most generally used logic programming language, often used in developing expert systems.

1972 Smalltalk, the first popular object oriented programming language, is developed at Xerox PARC by Alan Kay.

1972 C is created by Dennis Ritchie at Bell Labs for programming the Unix operating system. It is fast, widely used, and forms the basis of many other current procedural languages.

1975 Bill Gates and Paul Allen write the first version of Microsoft BASIC.

1978 Awk, a text-processing language named after the designers, Aho, Weinberger, and Kernighan, is developed for Unix.

1979 SQL (Structured Query Language) is developed at IBM based on work to simplify access to data stored in a relational database. It has become the most widely used database language.

1982 PostScript, a language for graphics printing and display, appears.

1983 C++, an object-oriented version of the C programming language, appears, based on earlier work on “C with Classes”. It is often used for large projects that require speed.

1986 Objective C, a Smalltalk influenced object-oriented version of C, became widely used as the development language for NeXTstep, and is currently the principle programming language for Mac OS X.

1987 Perl (Practical Extraction and Reporting Language) is developed by Larry Wall after he finds Unix text utilities limiting. It has become popular as a jack-of-all trades language, and in computational biology applications.

1991 Python, a simple functional and object oriented language, is developed by Guido Van Rossum. It is often used for rapid development, and is well suited for computational biology applications.

1991 Visual Basic is developed by Alan Cooper and Microsoft to allow for easy visual creation of Windows applications.

1995 Java, a simplified version of C++, originally developed by Sun Microsystems to control consumer appliances, is repurposed for web development. It has become popular for writing cross-platform and web applications.

1995 Ruby, a simple and elegant object oriented programming language, is developed by Yukihiro Matsumoto.

2002 C#, an object oriented programming language based on C++ and Java, is developed by Microsoft.

2014 Swift, an object oriented programming language for iOS and OS X, is developed by Apple.

Programming

Programming involves giving a series of instructions to a computer that tell it to perform a task. Programming languages allow one to communicate with a computer using source code that is closer to a natural language, such as English. There are three main types of programming languages: **Assembled, Interpreted** and **Compiled.**

Machine Code Programming
It is possible to program directly in the binary language of a computer (**0**'s and **1**'s). This is difficult and rarely done in modern programming.

Assembler
Automatically converts natural language into machine code. It is difficult and requires low level understanding of the machine, but can allow for the creation of highly optimized programs.

Interpreter
An interpreter translates the source code written in a particular programming language into the appropriate machine code as the program is run. The translation is done dynamically. This type of program is often called a "script". Perl, Python and Java are examples of interpreted languages (technically, their interpreters are interpreter/compliers). Many interpreted languages have an optional compiler.

Compiler
A compiler compiles the source code written in a particular programming language into executable machine code, creating a separate executable program which will always run as machine code. C is an example of a compiled language. Many compiled languages offer an interpreter as well.

Programming Languages

Macro
A single, user-defined command that executes a series of one or more commands (alias, Keyboard Shortcuts).

Scripting Languages
A simple programming language that uses a syntax close to a natural language and sends commands to the operating system or other programs when executed (AppleScript, bash, JavaScript).

Database Languages
A programming language tied closely to a database, allowing for easy queries and manipulation (SQL).

Procedural Languages
A fully featured programming language in which variables can keep changing as the program runs. Most commonly used programming languages are procedural (C, Perl).

Functional Languages
A fully featured declarative programming language based on the evaluation of mathematical functions in which variables do not change as the program runs (Erlang, Haskell).

Logical Languages
These programming languages are collections of logical statements and questions (Prolog).

Object Oriented Languages
A programming language in which data and functions are encapsulated in objects. An object is a particular instance of a class. Each object can contain different data, but all objects belonging to a class have the same functions or methods. Objects can restrict or hide access to data within them (C++, Objective C, Python, Java, Ruby).

Most Popular Programming Languages

1. Python
2. C
3. Java
4. C++
5. C#
6. Visual Basic
7. JavaScript
8. Assembly language
9. SQL
10. PHP
11. Objective-C
12. Go
13. Delphi/Object Pascal
14. MATLAB
15. Fortran
16. Swift
17. Classic Visual Basic
18. R
19. Perl
20. Ruby

Source: September 2022 TIOBE Programming Community Index. Note that the top five have not changed for 17 years except Perl, which was replaced by Python.

Commonly Used Programming Languages in Bioinformatics

C

The C programming language is one of the oldest programming languages still in wide use. A compiled C program offers excellent performance, and its syntax been very influential (**www.lysator.liu.se/c/**).

Python

A simple object oriented scripting language that is well suited for developing bioinformatics applications and available under a free open source license. It is particularly easy to read and understand, and has become increasingly popular in bioinformatics applications (**www.python.org**).

Java

Java is a powerful object oriented cross-platform programming language developed and made available for free by Sun. It was originally developed for controlling consumer appliances, but repurposed for web development, then expanded. It is simpler than C++, the object oriented version of C, but still take significant effort to master. It is very powerful, and has been used in a number of major bioinformatics projects (**www.java.com**).

R

R is a language for statistics, data visualization and data analysis. It is free and open source, and has become well established in data sciences and bioinformatics (**www.r-project.org**).

Perl

The Practical Extraction and Report Language (PERL) was once the most heavily used programming language in bioinformatics. It is distributed under a free open source Artistic License and became widely adopted by the open source programming community, resulting in numerous useful add on modules for Perl (**www.perl.org**).

Declarative or Imperative?

Declarative Programming

Declarative languages are less common, and describe what task a program should perform, without telling it how to perform the task, which the language handles.

Declarative languages can be domain-specific (such as SQL or HTML), functional (such as Haskell or R), logical (Prolog), mathematical, or hybrids.

Imperative Programming

Imperative languages are the most common, and use a series of statements, including control flow statements, to change a program's state, explicitly telling a computer what steps it should take.

Machine code and assembly languages function give instructions at a very low level. Procedural imperative languages (such as C) group sets of instructions into procedures. Object oriented imperative languages (such as C++, Java or Python) group instructions with the state they operate on.

Type System?

How a programming language handles the type of data used: booleans, integers, characters, dates, etc., depends on its type system. The more restrictions imposed by the language on changing type, the more **strongly typed** the language.

Static Typing

The language checks that a variable is always associated with a data type before the program is run. This value must be explicitly declared (C, Java) or can be inferred by the language (Haskell, Swift). Static languages that allow for unexpected type casting (C) are **weakly typed,** while those that do not (Java) are **strongly typed.**

Dynamic Typing

The data type associated with a variable is checked as a program runs (JavaScript Perl, Python, Ruby, R) and can vary dynamically. Dynamic languages that allow for an unexpected change in a variable type (JavaScript, Perl) are **weakly typed,** those that do not (Python, Ruby, R) are **strongly typed.**

Memory Management?

Manual Memory Management

Many older programming languages (C) require the programmer to manually allocate the memory a program will use, and then manually release the memory for reuse, which can be time consuming to implement, and a frequent source of bugs.

Automatic Memory Management

Most modern programming languages feature automatic memory management, the language will automatically allocate the memory required for the program to run, and automatic garbage collection to reclaim memory no longer required for use. There is a slight performance overhead, which is generally outweighed by substantial increases in programming speed and decreases in memory allocation bugs.

Batch Oriented or Event Driven?

Batch Oriented Programs

These are programs that are normally started from a command line (or run automatically by a scheduler such as cron). A batch program can simply consist of a text file with a list of programs it runs, or be more complex. When started, a batch program typically initializes the data inside it, reads in what data is specified as input, processes it, and outputs the result.

Event Driven Programs

There are programs that react to certain events sent to it by the operating system. This is typical of graphical user interfaces (GUIs), where an event might be a MouseUp (user moving the mouse up), or MouseClick (user clicks the mouse), which the program then responds to.

Object Oriented?

Object oriented programming (OOP) languages encapsulate **data** and **functions** in abstract data types called **objects.** Objects are designed in **class** hierarchies, and **inheritance** allow the data and functions in a class to pass down the hierarchy. Each object is a particular **instance** of a class.

Each object can contain different data, but all objects belonging to a class have the same functions. Objects can restrict or hide access to data within them. Functions in the object called **methods** are used to access data within that object. A class can be thought of as a template for an object, specified by a **class definition.**

Many popular non-object oriented programming languages (C, JavaScript, Perl) exist, but in recent years object oriented programming languages (Java, Python, Ruby, Swift) or languages that support object oriented use (C++, C#, Objective C) have grown in popularity.

Compiled or Interpreted?

Compiled

Compiled programming languages use a compiler to translate the instructions in the program into an executable program of machine code, which is then run.

Interpreted

Interpreted programming languages execute a program directly, with an interpreter translating each instruction into one or more subroutines precompiled into machine code.

Both

These distinctions have recently begun to blur, many interpreted languages now feature compilers, and many compiled languages now feature interpreters. Java features just-in-time compilation, in which a program is compiled as it is executed, which merges features of both approaches and allows for optimization such as dynamic recompilation.

Compiling a Simple C Program

1) Open a new document in BBEdit and type the following source code:

```
#include <stdio.h>
main ()
{
        printf("Hello world?\n");
}
```

2) Save the file in your home directory as **hello.c** (don't append .txt)

3) The source code must then be compiled to run. We can use the **gcc C** compiler to do this by opening Terminal, then typing `gcc hello.c` (on OS X this compiler and other optional command line tools can be installed in Terminal using the command `xcode-select --install`)

4) This creates a compiled executable program named **a.out** by default. Execute the newly created compiled program by typing `./a.out`

In the C programming language, we have to first compile our source code to an executable program (the compiler automatically set the permissions of the **a.out** file to be executable), then run the compiled program. Other programming languages such as Python or Perl are interpreted, which means that a text file containing source code for a Python script can be directly executed (assuming the text file has permissions set to allow it to be executed).

Configure, Make and Install

To install and compile a more complex Unix program, follow these steps:

1) Check for a file named **configure**. If one exists, run the command **./configure**. This will configure the installation for your system

2) Check for a file called **Makefile** or **make**. If it exists, run the command `make`. This will compile the program for your system. For simple programs, this may be the only step necessary

3) In some cases, you can test the compilation first by running the command `make test`

4) You may then need to finish the installation by running the command `make install`

If you have problems getting a Unix program to compile on OS X or cygwin, often all that is needed is a minor change to the text in the **configure** or **make** file. Occasionally you may need to run an additional command such as `make install-lib` or `run build` instead. Often these details can be found in the documentation or website for the program.

Installing a Unix C Program

Installing and Compiling a Bioinformatics C Program

1) Create **~/bin**

2) Copy **seqstat.tar** to **bin**

3) In Terminal, type `cd ~/bin` and press **return**, then type `tar -xvf seqstat.tar` to extract the files and press **return**

4) Type `cd seqstat` and press **return**, then type `ls` and press **return** to see what is there

5) Since a Makefile exists, simply type `make` and press **return** to compile the program

6) To run the compiled program, type `./seqstat` and press **return**. To quit a program while it is running, press **Control** and **C**

Interpreting a Python Script

1) Open a new document in TextWrangler and type the following source code:

```
#!/usr/bin/python
print "Hello, world?"
```

2) Save the file in your home directory as **hello.py**

3) Try to run the script by typing `./hello.py` and pressing **return**

4) Make any necessary modifications (you may need to use `chmod 755 hello.py`)

Note that in Python 3, `print` has become the `print()` function, so the object you wish to print must be wrapped in parentheses, e.g. `print("Hello, world?")`

Running an Interactive Python Script

1) Open a new document in BBEdit and type the following source code:

```
#!/usr/bin/python
your_name = raw_input('Enter your name: ')
print ('Hello, ' + your_name)
```

2) Save the file in your home directory as **ihello.py**

3) Try to run the script by typing `python ihello.py` and pressing **return**

4) Make any necessary modifications

Note that in Python 2.x `raw_input` is used to process a string, while `input` is used to process an expression. In Python 3, `input` processes strings, and `raw_input` no longer exists.

Structured Programming

In structured programming, programs are created using combinations of four constructs: (1) instruction sequences, (2) branches, (3) loops and (4) modules. The program uses these constructs to perform certain operations on data, which it can input and output.

Instruction Sequence
A sequential series of instructions.

Branch
A branch, also known as a conditional construct, occurs whenever a program's flow can divide into two or more streams, depending on whether a particular condition is true or false, such as whether a stop codon has been reached or not.

Loop
A loop repeats an instruction or series of instructions a variable number of times, which can be controlled by a test, such as whether the end of a DNA sequence has been reached.

Modules
Modules are a way to combine several operations (consisting of one or more of the other three constructs) into a single, reusable component. That component can then be reused throughout the program, or even used by other programs.

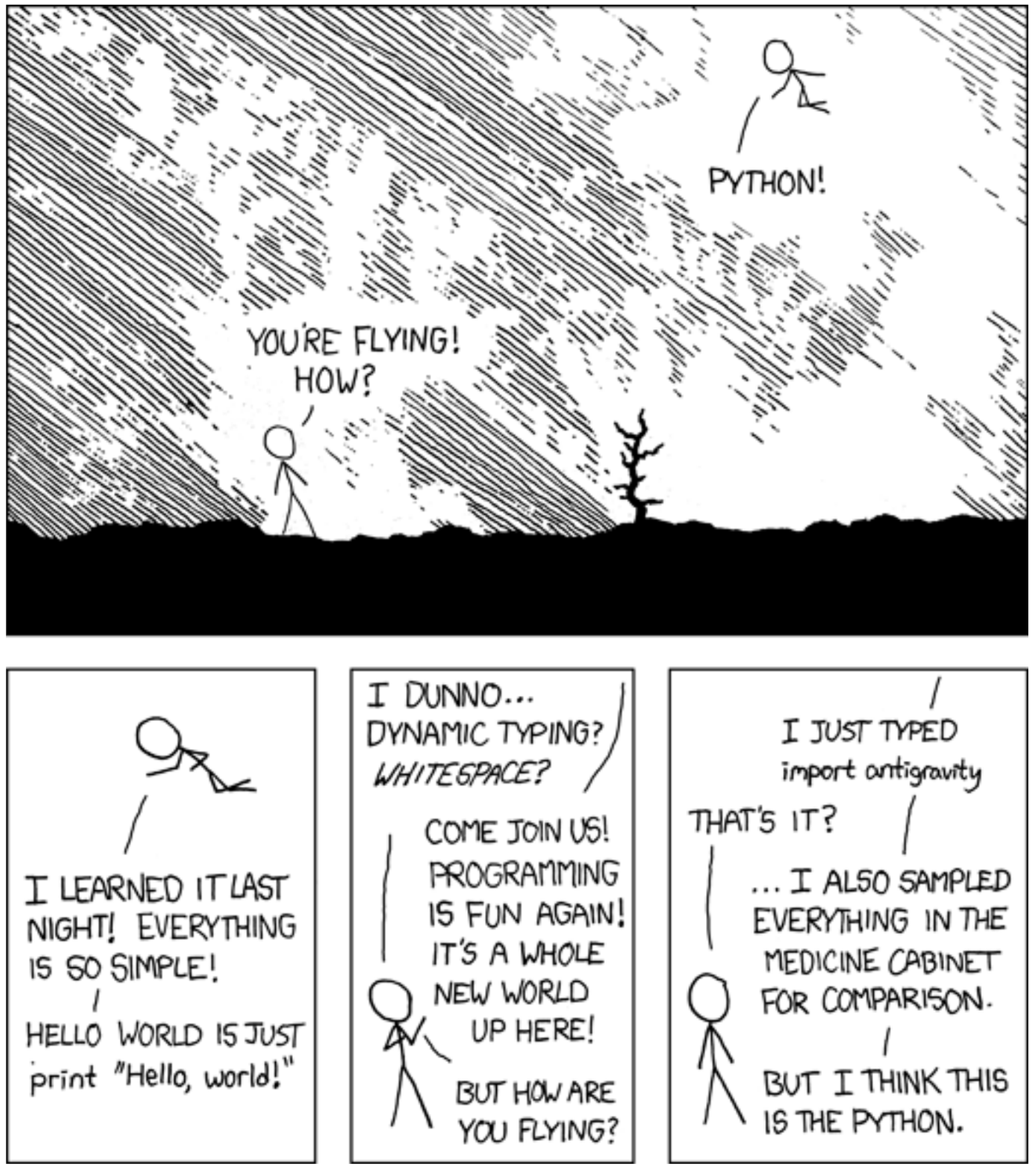
Python

The Python programming language was released in 1991 after two years of development by Guido van Rossum, a Dutch programmer, now considered the “benevolent dictator for life” of Python. The language’s name is a reference to Monty Python’s Flying Circus.

The reference implementation of Python is written in C and called CPython, and is free and open source, managed by the non-profit Python Software Foundation, and supported by a large community of open source developers. The Python Package Index (PyPi), which serves as a repository for free third party Python software, currently contains nearly 200,000 packages.

Python supports a variety of programming paradigms, including object-oriented, structured, functional and procedural programming. It is distinguished by its emphasis on simplicity and readability of code, and uses whitespace indentation to delimit blocks of code.

In recent years, Python has become one of the world’s most popular programming languages, used heavily at Google, Facebook, CERN and NASA, widely taught in introductory computer science courses, and is well established in bioinformatics.



Source: xkcd.com/353/

Instruction Sequences in Python

Instruction sequences in Python consist of statements and expressions.

Simple statements are written one per line, and semi-colons can be used to separate multiple statements on a single line. Statements generally perform some action, e.g. `return`, and can produce a value as a result, although statements to which a value is assigned (e.g. `x = 1`) will not directly produce a result. Commonly used statements include `def`, `break` and `return.`

Expressions consist of a combination of values, variables and mathematical operators that produce at least one value (e.g. `y = x + 1`). Even a lone value or variable can be considered an expression. Functions are code to which parameters can be passed to return a value, e.g. `range(1,11)`. Blocks of Python text that are to be executed as a function are delineated by indented white space. When the indent ends, the function ends. Note that in most other programming languages, curly braces `{}` are used for this purpose.

The # symbol is used to indicate a comment. Anything on a line after a # symbol is ignored by Python.

Variables in Python

Variables are one of the most useful features of programming languages, allowing a name to be associated with a stored data value, such as a string of text or a number, that can change as the program runs. Python does not require variables to be explicitly declared, and can handle variable types including: **number, string, list, tuple** and **dictionary.** The `type` statement will identify the type of a variable.

Number: A number type can be an integer, long integer, float or complex number, e.g. `int_value = 7`

String: A string can be delimited by single, double, triple single or triple double quotes and can contain tab or newline characters. Strings are immutable, functions return new strings derived from the original, e.g. `dna_sequence = "GCATTTGTGAGACCCCGTACGTAG"`

List: A list holds multiple values of different types in an ordered list of data, e.g. `rna = ["G", "C", "A", "U"]` The first element in an list is numbered 0, the second 1, the third 2, etc, and a value can be retrieved by specifying its position, e.g. `rna_value = rna[3]`

Tuple: A tuple is similar to a list, but immutable, and are generally used to provide keys for dictionaries.

Dictionary: A dictionary acts as an associative array, associating an immutable key with any kind of value, e.g. `stop = {"amber":"AUG", "ochre":"UAA", "opal":"UGA"}`. The value can then be retrieved from the dictionary using the appropriate key, e.g. `codon_value = stop["amber"]`

Branches in Python

if-elif-else

The block of code after the first true condition of the **if** clause or any number of optional **elif** clauses is executed. If none are true, and an optional **else** clause exists, the block of code following the **else** is executed. In Python, blocks are indicated by indentation. Each block consists of one or more statements separated by new lines at the same level of indentation, e.g.

```
if dna_length > 1000:
    algorithm_to_use = "longblast"
elif dna_length > 100:
    algorithm_to_use = "midblast"
else:
    algorithm_to_use = "shortblast"
```

There is also a one line syntax that allows for simple conditional expressions (if-else): [on_true] **if** [expression] **else** [on_false], e.g.

```
number = input("Enter a number for absolute value: ")
print (-number if number < 0 else number)
```

For Loops in Python

for

The **for** loop can iterate over any items in a list or tuple. A **break** statement can be used to end the loop after it finds what you are looking for. A **for** loop is also used when you want to repeat something **n** times, e.g. for ten times:

```
for x in range(1,11):
    print "x is now %d" % (x)
```

Note that in most other programming languages, particularly those derived from C, a **for** loop will look something like this:

```
for (i = 1; i < 11; ++i)
{
    printf("%d ", i);
}
```

While Loops in Python

while

The **while** loop is executed as long as the condition is true:

```
i = 1
while i < 11:
    print(i)
    i += 1
```

A **break** statement can be used to end the loop. It is often used to process input in Python, e.g.

```
while True:
    x = raw_input("Please type goodbye:")
    if n.strip () == 'goodbye'
        break
```

Defining a Function in Python

You can easily define (**def**) your own functions in Python.

```
#!/usr/bin/python
def algorithm(dna_length):
    if dna_length > 1000:
        algorithm_to_use = "longblast"
    elif dna_length > 100:
        algorithm_to_use = "midblast"
    else:
        algorithm_to_use = "shortblast"
    return algorithm_to_use

#main
user_input = int(raw_input("Enter DNA length: "))
print algorithm(user_input)
```

Python Input and Output

Keyboard input to a Python program can be obtained using the **raw_input** function, which returns whatever the user typed up to pressing **return** as a string (or **input** for an expression, though note that this syntax differs in Python 3, where **input** returns a string).

Opening a file for reading or writing is done using the **open** function. By default, files (technically file objects) are opened for reading, specifying **'w'** opens files for overwriting (any existing data in the file will be erased), **'a'** opens files for writing in append mode (new data is appended to data already in the file). Once finished reading or writing, the file should be closed using the **close** method.

The **readline** method reads a single line including any newline character, and is commonly used to read a file a line at a time, while **readlines** reads all the lines in a file, and returns them as a list of strings, one per line. The **write** method writes a single string (which can include newline characters) to a file, while **writelines** writes a list of strings to a file.

```
file_object = open("anybody.txt", 'w')
file_object.write("Is there anybody out there?")
file_object.close()
```

Modules in Python

Python is fundamentally a modular language. Complex programs are often split into modules for ease of maintenance and reusability.

A Python module is simply a text file containing additional definitions and statements. The filename should end in **.py** and the name of the module (the filename) is available in the module as the value of the global variable **_name_**. Packages are organized collections of modules (in other langauges, they may be called libraries.)

Python comes with a number of default modules that are already installed as part of its standard library, such as the **string** module, which supports common string operations, or the **sys** module, which allows access to command line arguments passed to a script. To add functionality from such a module, it has to be imported using the syntax `import modulename` or `from modulename import *`.

It is possible to install and then import many modules or packages, including third party libraries such as BioPython, which add significant functionality to Python. Be aware that third party modules may have dependencies, that is they may need other third party modules or packages (e.g. NumPy or SciPy) installed to function.

Using a Python Module

randomdna.py

This Python script to generates a random sequence of nucleotides of length 100, using the `random` module, the functions `join`, `random.choice` and `xrange`, and a `for` loop with a temporary `_` variable and a `length` variable:

```
#!/usr/bin/python
import random
def DNA(length):
    return ''.join(random.choice('acgt') for _ in xrange(length))
print DNA(100)
```

For your take home assignment, you will be modifying **randomdna.py** to function more interactively, writing a specified length of random DNA to a specified file name.

Writing to a File With a Python Script

anybody.py

This Python script writes a string to a file called **anybody.txt** (overwriting it if it already exists).

```
#!/usr/bin/python
file_object = open("anybody.txt", 'w')
file_object.write("Is there anybody out there?")
file_object.close()
```

For your take home assignment, this approach can be used for modifying **randomdna.py** to write to a file. For modifying **countgc.py** to read from a file, you may want to look into the read option of the `open` function, or look into the `read` function.

Counting %GC with a Python Script

countgc.py

Uses the string library to count percent GC from `raw_input` (or `input` in Python 3).

```
#!/usr/bin/python
from string import *

def count_gc(dna):
    count_g = count(dna, 'g')
    count_c = count(dna, 'c')
    dna_length = len(dna)
    percent_gc= 100 * float (count_g + count_c) / dna_length
    return percent_gc

dna = raw_input ("Enter a lowercase DNA sequence: ")
print count_gc(dna), "percent GC"
```

For your take home assignment, you will be modifying **countgc.py** to function more interactively, reading and counting percent GC from a specified file.

Programming References

Think Python: How to Think Like a Computer Scientist by Allen B. Downey, free at **http://www.greenteapress.com/thinkpython/thinkCSpy.pdf** with an interactive website at **https://runestone.academy/runestone/books/published/thinkcspy/index.html**

The Python Tutorial free at **https://docs.python.org/3/tutorial/index.html**

Dive Into Python 3 free at **https://diveintopython3.problemsolving.io**

Learn Python the Hard Way at **https://learnpythonthehardway.org/book/**

The Quick Python Book, 3rd Edition by Naomi Ceder

Python Programming: An Introduction to Computer Science, 3rd Edition by John Zelle

Bioinformatics with Python Book Cookbook, 2nd Edition by Tiago Antao