Comparative Programming Languages
CM20318

Russell Bradford

2023

Introduction

The purpose of this Unit:

How to choose the right tool for the job

Or…

Why *insert favourite language here* is not always the answer

But sometimes is

“Why are you teaching me how to use a screwdriver when I am an expert in using a hammer?” (paraphrase of a student question)

CM20318

Three hours of lectures a week:

- Monday 16:15
- Wednesday 12:15
- Thursday 10:15

The plan is to cover the content early, leaving a few weeks at the end for revision classes

Introduction

The lectures will be recorded (to Re:View/Panopto)

If you have a personal issue with being recorded, please sit in a place not covered by the lecture room camera

CM20318

Assessment is 100% exam

This unit was new in 2022

It is based on a half-sized unit named CM20253

See past papers for CM20253 online for representative questions, though the style of the CM20318 paper is slightly different

The papers for 2021/2022 and 2022/2023 are available in the usual place

CM20318

Learning Outcomes:

On completion of the unit the students will be able to:

1. recognise the various styles of programming language
2. understand the differences between them
3. choose the right programming style and language for the task in hand.

Styles

Content:

Programming paradigms and language families, for example Functional, Procedural, Object Oriented, Logic, Scripting, Declarative, Macro, Unstructured, Event Driven, etc. Examples of languages from each style, with comparisons.

Further choices that languages provide, e.g., application based languages; interpreted, bytecoded and compiled; parallel or sequential; OO prototyping, delegation, traits, class centred; managed, unmanaged and garbage collected; static typed, dynamic typed and untyped; call by value, reference, name, need, etc.

CM20318

I may include examples of code, without much description, in a language you have not seen before. You should

- not panic
- use this as an opportunity to go and read about that language for yourself

But, most importantly, you should be getting to the stage of seeing unfamiliar code and being able to have a good guess at what it does

Some details may need researching, but the general idea should be clear

Introduction

The main idea of this Unit is to show you many kinds of language and many kinds of programming, to equip you with the means to make the choice of the right tool for the job

So you don’t try to solve every problem with a Java- or Python-shaped mallet

Introduction

But this Unit is *not* about teaching you new languages

It about exploring programming *concepts*

Even if you choose to use a familiar old language, you can use it more effectively in knowledge of these concepts

Would you hire a carpenter whose only tool is a screwdriver?

Introduction

Even if you are a stuck-in-the mud Java (for example) programmer and will use Java forevermore, there are many things in this unit you need to know

Some constructs (e.g., iterators, lambdas) that Java is adopting; also some concepts (e.g., automatic type inference)

Understanding (a) what they do (b) why they exist (c) why Java is adopting them (d) how to use them effectively; all these will make you a better Java programmer

And if you end up being a manager rather than a programmer, it helps to know what languages or features in languages are there to help weaker (= cheaper) programmers be more productive

Introduction

Note that some people get *very* wound up by some of the subjects that we will cover

A lot of flame wars on the Internet forums dealing with programming or programming languages are “programming language X is good” vs “programming language X is bad” where they really mean “X is really good for what I do” vs “X is really bad for what I do” but the “for what I do” is lost and so the arguments go on forever

Introduction

People often get stuck in the mind-set of the first language they learned, e.g., Java or Python these days. They try to approach all problems with the tools/concepts they have available in that language. When a new language comes along with a new approach, often their instinct is to say the new language is defective (as it doesn't do things in the old way) and rubbish it and not try to learn the new ideas it presents

Introduction

“No, not in any other language, just the very few you happen to have encountered so far.
It’s really important not to fall into the trap of believing that the first language you encounter is normal, and anything you meet later, that differs from it, is weird. Falling into this trap is a huge disservice to yourself, as it will make learning new things much more frustrating and much less productive for you”

jacg, in reply to a comment in a language forum on how that language “gets it wrong” as “all other languages” do something different; July 2023

After a few examples of different ways languages do things, jacg continues:

Introduction

“If you think that some of them are more logical or natural or correct than others, then you are deeply mistaken. Do yourself a favour and disabuse yourself of this misconception! Understand that the syntax and the concept the syntax represents are separate things, and learn to understand the concept in itself, liberating yourself from the syntax. That way, a number of important and hugely beneficial to you, things will happen:

– You’ll have a far firmer, deeper, better understanding of the meaning of programs you read and write.

– You’ll be able to reason more clearly about the ideas you want to express, because you will be able to decouple thinking about what it is that you want to say, from thinking of how to say it in whatever language you happen to be using.

– You’ll be able to learn new languages far more efficiently.”

Introduction

"Hidden within the various languages that exist today, are *a set of paradigms that can completely change the way you are used to thinking*. Sometimes these paradigms are so focused and so specific to a language that they are only applicable in that particular language. Other times I find, and this is the great part; that you can **take those paradigms and apply them to the languages you currently utilize**. When that happens, congratulations, you've expanded your mind and your skill set and additionally you now have a fresh way of tackling stale old problems."

Ralph Caraveo III

Introduction

“Reading and experience train your model of the world. And even if you forget the experience or what you read, its effect on your model of the world persists. Your mind is like a compiled program you’ve lost the source of. It works, but you don’t know why.”

Paul Graham

Introduction

> “If your boss was taking all the decisions for you, you wouldn’t be a programmer, you’d be a typist.”

“Bloggy Badger”

> “Programming is a skill acquired by practice and example rather than from books.”

Alan Turing

> “Programs must be written for programmers to read, and only incidentally for machines to execute.”

Abelson and Sussman, “Structure and Interpretation of Computer Programs”

Introduction

“You cannot reduce the complexity of your problem by increasing the complexity of your language”

Niklaus Wirth

“There are two ways of constructing a software design: one way is to make it so simple that there are obviously no deficiencies, and the other is to make it so complicated that there are no obvious deficiencies.”

Tony Hoare

Resources

A lot of the material I shall cover is “mature”, meaning it’s been around for a long time

So there are lots of books on programming languages out there and to some extent it’s a matter of finding a book that suits you

The number of books on *comparative* programming languages is quite small

E.g., “Comparative Programming Languages” by Wilson and Clark, 3rd Edition

Though this is a bit lightweight

Resources

There is a unit Web page that contains bits and pieces relevant to this unit:

`http://people.bath.ac.uk/masrjb/CourseNotes/cm20318.html`

Web-based material tends to be fairly accurate: Wikipedia is pretty good in this area

But, as always with Wikipedia, you should treat it as only the start and *follow up the references*

Exercise Have a look at `http://rosettacode.org/`

Resources

Contacting me:

If you have a question on the unit, please consider bringing it along to the revision classes so that everyone can get the benefit

Otherwise, email me — I don't monitor all the dozens of other ways of messaging (Moodle, Teams, etc.) and email is the only way to be sure of getting a message to me

I keep a 9-5 (approx) Monday–Friday week and am unlikely to respond out of those times (a long time a ago someone said "Get a life", so I did)

Material

Note that this Unit covers a **lot** of many different things

After all, its purpose is to expose you to things that you possibly wouldn’t normally come across

Therefore it can be quite light on detail in places

Continuing Exercise Read around the topics to fill in more detail

Continuing Exercise When you come across a new (to you) language, write some code using it

Material

As this is an overview Unit that covers many different topics, your background reading (which is expected of every Unit) is particularly important to help you consolidate what we are going to see in lectures

Standard Introductory Slides

Remember:

You are expected to do some work outside of lectures

Lectures are the *start* of the learning process, not the end!

These slides are reminders to me on what to say in lectures

They are often abbreviated in style, and so are not the whole story and would not be suitable to be quoted verbatim in an exam

Standard Introductory Slides

Don’t try to copy everything down from the slides in lectures—the slides will be available after each lecture

Instead, make a note of what is important and use that later—in conjunction with the slides—to guide your further reading and study

Standard Introductory Slides

Do not rely purely on my notes for your revision

People who do this live to regret it

Like every Unit, you are expected to read around the subject for yourself

You need to take your own notes, read, and *participate*

You don't expect to get fit simply by paying to joining a gym...

"If you have college courses in CS, buy the books and spend day and night the few days before class going through the books and taking notes and answering questions and programming examples before the first class even starts. If you really want to do this in your life, that's what you should do, not just wait for the education to be handed you. Those who finish at the top will always be in high demand. You can learn outside of school too but you have to put a lot of time into it. It doesn't come easily. Small steps, each improving on the other, is what to expect, not instant understanding and expertise."

Steve Wozniak, co-founder of Apple

Standard Introductory Slides

Computer Science is not a spectator sport

Anon