Back to the University

Diving into Scientific Programming

Willem L. Middelkoop
Feb. 8, 2023

This month, I headed back to university for a special course on scientific programming. With numerous recent advancements in machine learning (referred to as "AI" in PowerPoint parlance), including ChatGPT, it felt like the ideal moment to delve into Python programming. Join me on this journey.

Why Python for Scientific Programming?

Python has gained immense popularity within the scientific community, and for good reason. Its simplicity and readability make it approachable for newcomers, whilst retaining the power to tackle complex tasks. The language's versatility enables researchers and scientists to implement algorithms and models with relative ease.

An extensive array of libraries tailored for scientific computing, such as NumPy, SciPy, Pandas, and Matplotlib, bolsters Python's position in this domain. These li-

braries offer comprehensive functions and tools for data analysis, statistical modelling, and visualisation, thus streamlining the scientific programming process.

In comparison to other programming languages, such as JavaScript or Ruby, which often find use in non-scientific contexts like web development, Python boasts a more straightforward syntax. This characteristic allows users to focus on the problem at hand rather than wrestling with language intricacies. Additionally, Python's extensive scientific ecosystem renders it a more suitable choice for research applications.

Examine the following two code samples, one in C++ and the other in Python 3. Each program performs identical tasks: requesting the user's name and date of birth, carrying out basic input validation, and ultimately revealing the user's current age.

```cpp
#include <iostream>
#include <string>
#include <sstream>
#include <ctime>

bool validate_date(const std::string &date_string, int &day, int &month, int &year) {
    std::istringstream iss(date_string);
    char delimiter;
    if (iss >> day >> delimiter >> month >> delimiter >> year) {
        return true;
    }
    return false;
}

int calculate_age(int day, int month, int year) {
    time_t now = time(0);
    tm *ltm = localtime(&now);

    int current_year = 1900 + ltm->tm_year;
    int current_month = 1 + ltm->tm_mon;
    int current_day = ltm->tm_mday;

    int age = current_year - year - ((current_month < month) || (current_month == month && current_day
< day));
    return age;
}

int main() {
    std::string name, dob_string;
    int day, month, year;

    std::cout << "Please enter your name: ";
    std::getline(std::cin, name);

    do {
        std::cout << "Please enter your date of birth (dd-mm-yyyy): ";
        std::getline(std::cin, dob_string);

        if (!validate_date(dob_string, day, month, year)) {
            std::cout << "Invalid date format. Please try again." << std::endl;
        }
    } while (!validate_date(dob_string, day, month, year));

    int age = calculate_age(day, month, year);
    std::cout << name << ", your age is " << age << " years." << std::endl;

    return 0;
}
```

C++ example code: Determining the user's age

```python
from datetime import datetime

def get_user_input(prompt):
    while True:
        user_input = input(prompt).strip()
        if user_input:
            return user_input

def validate_date(date_string):
    try:
        date_obj = datetime.strptime(date_string, '%d-%m-%Y')
        return date_obj
    except ValueError:
        return None

def calculate_age(dob):
    today = datetime.now()
    age = today.year - dob.year - ((today.month, today.day) < (dob.month,
dob.day))
    return age

name = get_user_input("Please enter your name: ")
dob_string = ""

while not dob_string:
    dob_string = get_user_input("Please enter your date of birth (dd-mm-yyyy): ")
    dob = validate_date(dob_string)
    if not dob:
        dob_string = ""
        print("Invalid date format. Please try again.")

age = calculate_age(dob)
print(f"{name}, your age is {age} years.")
```

Python 3 example code: Determining the user's age

One can easily observe the contrasting syntax between the two languages, with Python bearing a closer resemblance to everyday English. This aspect of Python often enhances readability and comprehension for many users.

Another factor contributing to Python's success in scientific programming lies in its thriving community. Researchers and developers from various disciplines share knowledge, offer support, and contribute to open-source projects, fostering an environment of collaboration and continuous learning.

The Course

Throughout the course, the absolute basics of programming were explored by tackling problems from various scientific domains. Upon completion, a solid grasp of programming principles was acquired, with the ability to apply them across different fields and projects.

The Universiteit van Amsterdam at the Science Park campus - friends of the blog will recognise my bike

Revisiting the basics of programming, even for experienced programmers (like myself), can prove to be immensely beneficial. Over time, it's not uncommon for programmers to develop habits or shortcuts that may not adhere to best practices. By returning to the fundamentals, you can unlearn any detrimental habits and re-establish a strong foundation in programming principles. Additionally, the fast-paced nature of the tech industry means that innovations and updates are constantly emerging. By revisiting the basics, seasoned programmers can stay up to date with the latest advancements and methodologies. This process of re-evaluation also offers an opportunity to approach programming with a clean slate.

Me taking part in a programming class at the University (class mates blurred for privacy reasons)

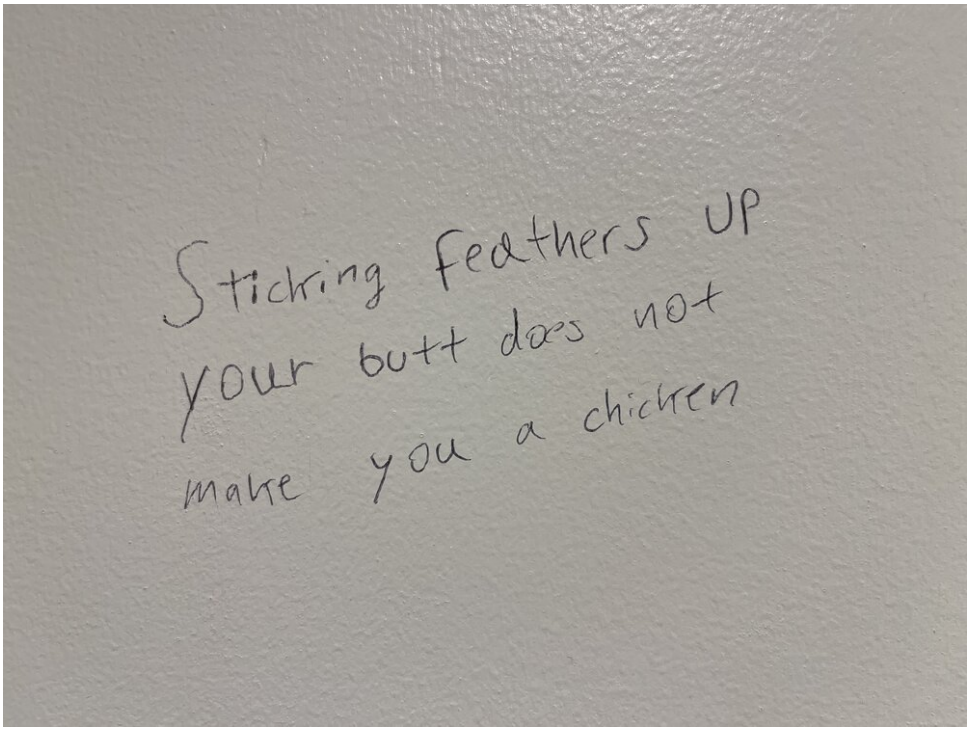
The course consisted of multiple modules, starting with Level 1, where a choice was made between ALGORITHMS, which focused on breaking down intuitive problems into steps a computer could understand, and NUMBERS, a mathematically oriented module that delved into number properties without requiring prior maths knowledge. In Level 2, the choice was between TEXT, which centred on natural language processing and sentiment analysis of tweets, and NUMERICAL INTEGRATION, where important techniques for determining surface areas under functions when traditional integration didn't suffice were taught. Level 3 involved working with BIG-DATA, analysing large datasets and weather patterns in the Netherlands. An optional bonus level, MOVEMENT, provided a simulation of digging a tunnel through the planet and explored physics problems solvable with computer assistance.

The weekly lab sessions offer a delightful opportunity to stay on track with your coding adventures, ensuring you have a helping hand from others whenever you hit a snag.

Bang on trend with a bleeding-edge gender-neutral toilet

Conclusion

No matter one's age or wisdom, there's always room to learn something new, particularly with advancements in machine learning like GPT4. Moreover, going back to university provides a refreshing change, presenting a completely different context compared to the commercial or professional sphere in which I typically operate. Embrace the mantra: stay hungry, stay foolish!



Toilet wisdom