Python Programming for Chemists: Introduction to Computing

Christoph Loschen

WS 2024/2025

Lecture 1.1:
Computing in a Nutshell

Lecture 1.1: Computer Basics

Computers & algorithms are used in many areas of chemistry:

- Analytics
  - Examples: X-ray crystallography, IR - spectroscopy, chemometrics
- Modeling & simulation
  - Examples: Computational Chemistry, Catalyst Development
- Visualization & data analysis
  - Plotting, statistics, machine learning
- Chemical databases
  - Examples: PubChem, Chemical Vendors (Merck), Pharma companies

Computers & Chemistry

Some Noble Prize Winners

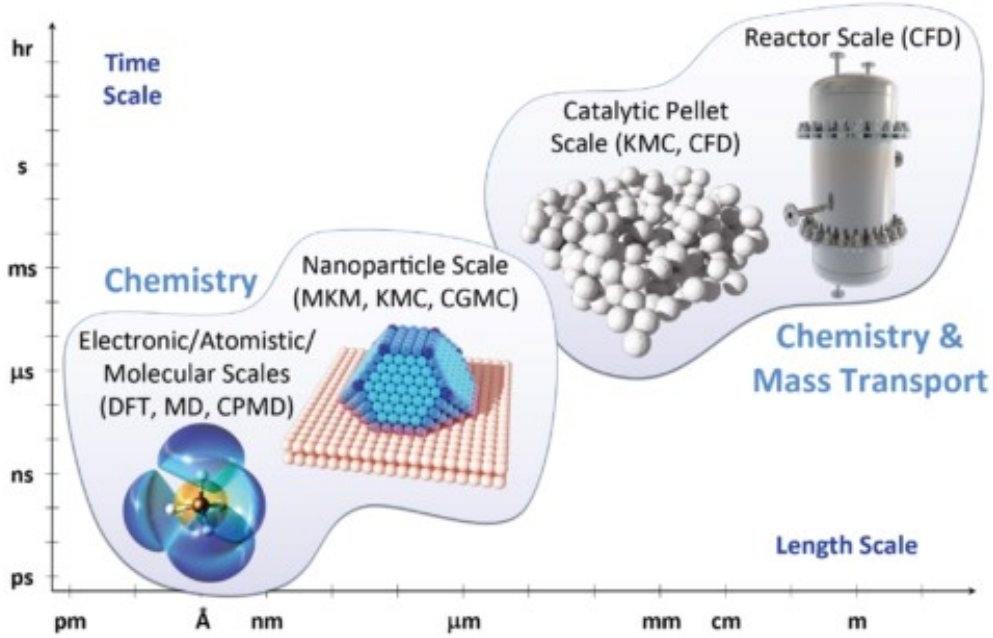
- Kenichi Fukui, Roald Hoffmann (1981):
  Understanding the reactivity of molecules, MO theory

- Walter Kohn, John A. Pople (1998):
  Density-functional theory (DFT) and quantum chemistry

- Martin Karplus, Michael Levitt, and Arieh Warshel (2013):
  Multiscale models for complex chemical systems

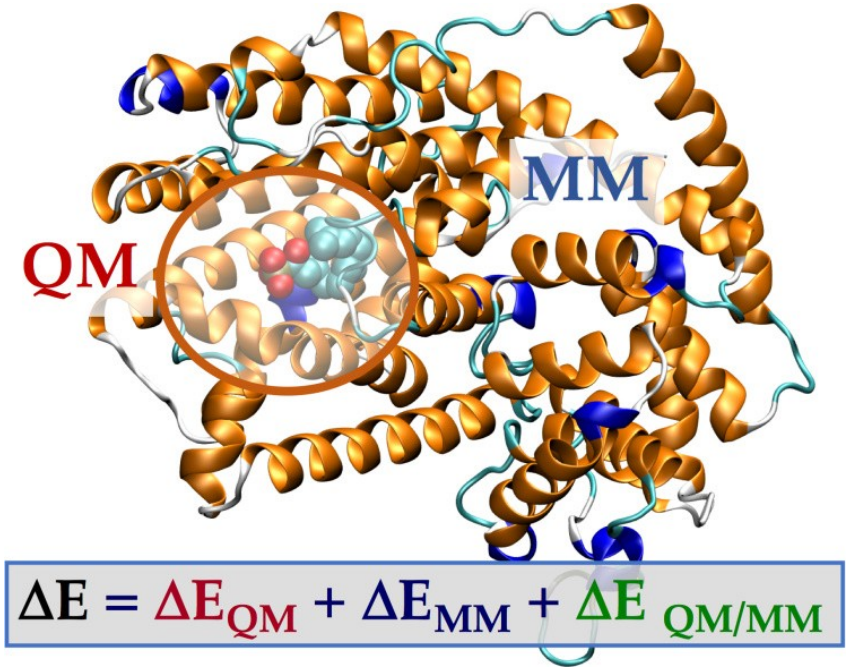
Computers & Chemistry

Multiscale Nature of Materials



Computers & Chemistry

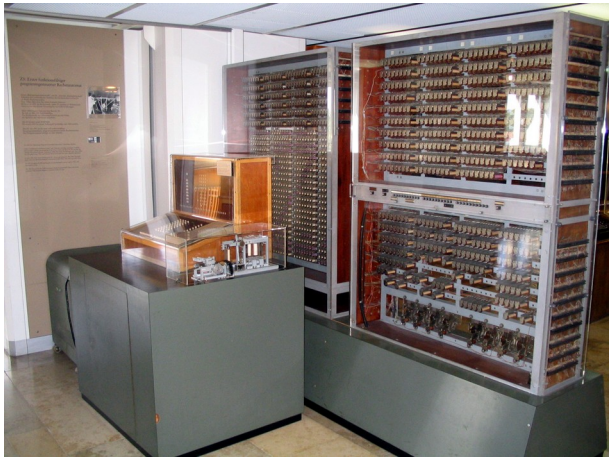
- Multi scale Example QM – MM
- Different areas in a protein can be computed with different methods: Quantum physics/ Classical physics



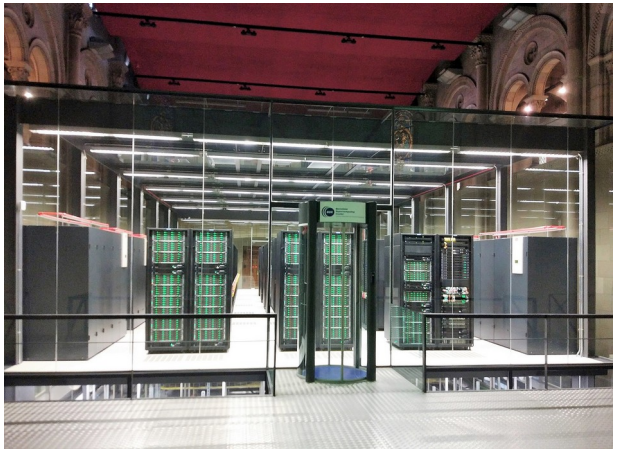
- We need computers and algorithms do help us understand the complexity of materials!

Introduction to Computers

- A computer is an electronic device that can store, process, and retrieve data. It follows a set of instructions (program) to perform a wide array of tasks.



Z3 (Konrad Zuse, 1941)



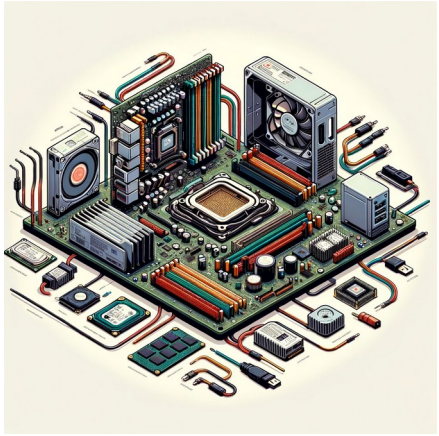
Mare Nostrum (2017)

Cell phone (2024)

Hardware vs. Software

- Hardware (physical components of a computer like the motherboard, CPU, RAM) – what we can touch
- Software (programs that perform tasks, such as word processors and games, “apps”) – what we see on the screen

Hardware



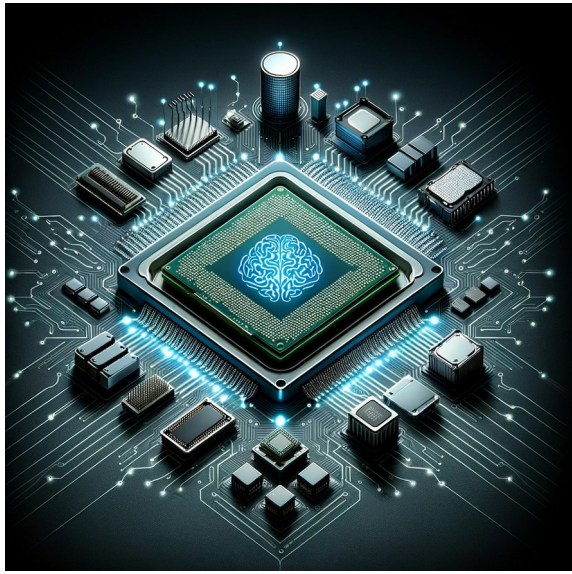
Software

These images were created with the assistance of DALL·E 2

The Central Processing Unit (CPU)

The Brain of the Computer - Central Processing Unit

- CPU is handling calculations and processing data
- Communicates with all other hardware component
- Decides how fast calculations can be done
- In graphics card: “GPU” (graphical PU)
- Parallel processing: Speed up by splitting tasks on several CPUs



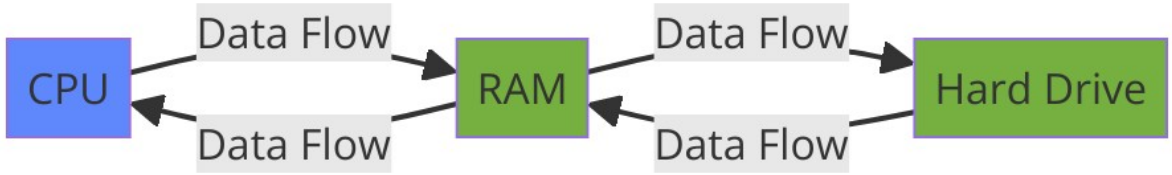
These image was created with the assistance of DALL·E 2

Graphical Processing Units

- A GPU is a specialized processor designed to accelerate graphics rendering and image processing.
- Commonly used in systems where high graphical performance is required: gaming, video editing, and AI computations.
- GPUs are increasingly used for machine learning, deep learning, and scientific simulations
- Mathematical operations for video graphics rendering and machine learning are similar: vector & Matrix operations, linear algebra, parallel processing

Computer Memory and Storage

- RAM (Random Access Memory) is storing *smaller* data *temporarily* while the CPU processes it (fast)
- Hard drive (disc) used storing *large* data permanently (slower)
- Usually data is loaded from the hard drive to the RAM to be processed by the CPU
- Size of data measured in GigaByte (25 GB ~ high resolution movie)



Data & Sizes

- Comparing different data storage devices and their sizes (2023)

Data source	Size (GB)
Book	0.001
Movie/Blue ray	5
RAM	32
Hard disc	~1000
DNA (1g)	215,000,000
Internet	120,000,000,000,000

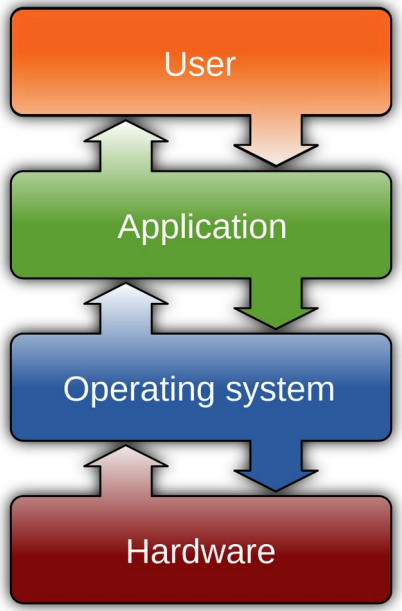
- Data is the new oil / raw material / gold
- How do you take care of your personal data?

Operating System and Applications

- Operating systems (OS) manage the computer's basic functions and serve as a platform for running various applications.
- Example: Windows, Linux, MacOS, Android, iOS

Tasks

  - Process management, Interrupts
  - Memory management
  - File system, Input/output
  - Device drivers
  - Networking, Security



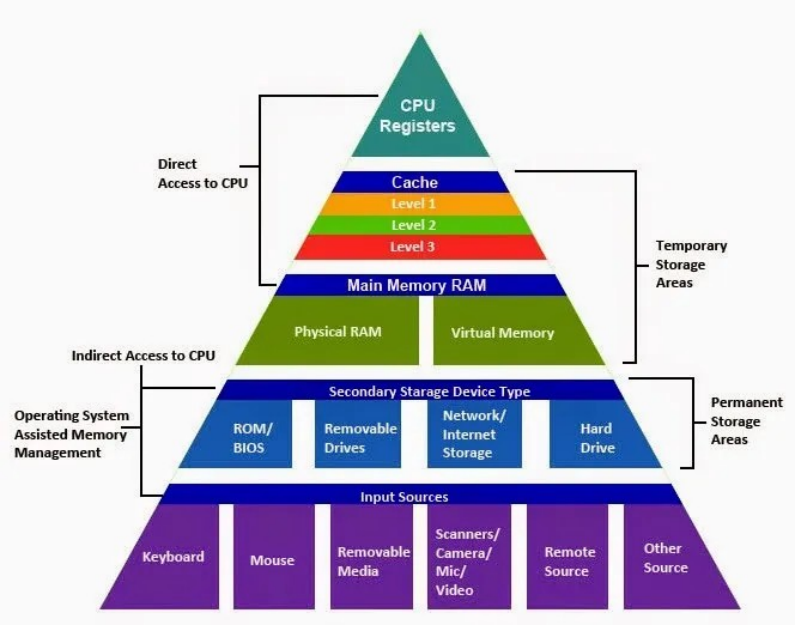
https://en.wikipedia.org/wiki/Operating_system

How does it work together?

- CPU Registers:Top level and fastest type of memory. Utilized for immediate operations and execution.
- Cache Memory: Exists in multiple levels (L1, L2, L3) as intermediate storage to reduce data access time.
- Main Memory (RAM):

  Primary storage for currently running programs and data in use.
- Secondary Storage:

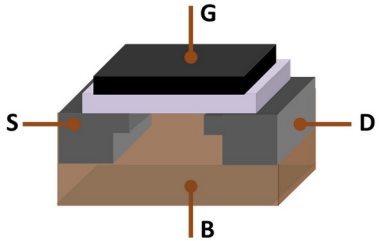
  Includes ROM/BIOS, removable drives, network storag and hard drives.
- Input Sources



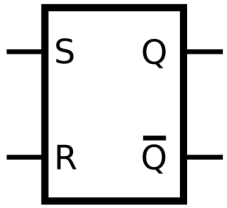
https://tvtropes.org/pmwiki/pmwiki.php/UsefulNotes/MemoryHierarchy

Transistors

- Basic components to modern electronic devices: computers, smartphones, and digital circuits.
- Small electronic components that can act as amplifiers or switches.
- Made from semiconductor materials like silicon or germanium.
- Can be used for so-called “Flip-Flops” e.g. as static 1Bit memory (on/off or 1/0)



Transistor



FliFlop

https://en.wikipedia.org/wiki/Transistor
https://en.wikipedia.org/wiki/FlipFlop

The Age of Computational Materials

- Stone age
- Bronze age
- Iron age
- Industrial age
- Plastic age
- Silicon age
- AI / data / material design age?

Lecture 1.2:
Progamming Basics

Example 1

- Row 1 (RS): K5, *k3, k2tog, yo, ssk, k3; work from *, k5.
- Row 2 (WS): P5, *p4, (k1, p1) in 1 stitch, p4; work from *, p5.
- Row 3: K5, *k1, LLI, (k2tog, yo, ssk) x 2, RLI, k1; work from *, k5.
- Row 4: Purl.

Example 2

Quick Lasagna Recipe

Ingredients:

- 12 lasagna noodles
- 2 cups ricotta, 1 cup parmesan (mixed)
- 4 cups mozzarella
- 5 cups marinara sauce

Instructions:

- Cook noodles and mix ricotta + parmesan.
- Layer in a dish: noodles, sauce, ricotta mix, mozzarella. Repeat 3 times.
- Bake at 300°C for 50 minutes (covered), then 10 minutes uncovered.

Example 2

```
// Ingredients
noodles = 12;
ricotta = 2;
mozzarella = 4;
parmesan = 1;
sauce = 5;

// Functions
ricottaMix = mix(ricotta, parmesan); // Prepare ricotta mix

// Layer and bake
repeat (3 times) { add(noodles, sauce, ricottaMix, mozzarella); }
bake(50 mins, 573K, covered);
bake(10 mins, uncovered);
```

Learning Programming

Learning programming is like learning cooking or knitting and consists of:

- A regular *vocabulary* of words, abbreviations and symbols
- Rules about the order and arrangement of words: *syntax*
- *Grammar* includes meaning of words, internal structure and syntax
- A *sequence of operations* to be performed in order
- *Repetions* of operations (loops)
- *References* to another sequence of operations (functions)
- Assumed knowledge about a *context*
- *Data* and *tools*
- An expected *result*

Progamming Basics

- A programming language is a **formal language*** comprising a set of instructions to communicate with computers
- Used to create software to perform complex tasks, automate processes, and solve problems efficiently.

Types of Programming Languages

- Low-Level Languages: Close to machine code, such as Assembly Language, offering fine control over hardware but requiring detailed knowledge
- High-Level Languages: More abstract and closer to human language, like Python, Java, JavaScript designed for ease of use and understanding

*Formal language: https://en.wikipedia.org/wiki/Formal_language

Lecture 1.2: Progamming Basics

source code examples

asm

```asm
section .data
    hello db 'Hello, World!',0xa  ; 'Hello, World!' plus a linefeed cha
    helloLen equ $ - hello        ; Length of the 'Hello, World!' stri

section .text
    global _start                 ; Linker needs this to find the entr

_start:
    mov eax, 4                    ; The syscall number for sys_write (
    mov ebx, 1                    ; File descriptor 1 is stdout
    mov ecx, hello                ; Pointer to the string to be printe
    mov edx, helloLen             ; Length of the string to be printed
    int 0x80                      ; Call kernel

    mov eax, 1                    ; The syscall number for sys_exit (1
    xor ebx, ebx                  ; Return a code of 0
    int 0x80                      ; Call kernel
```

c

```c
#include <stdio.h>

int main()
{
    printf("Hello, World! \n");
    return 0;
}
```

python

```python
#!/usr/bin/python3

print("Hello, World!")
```

Low level → high level

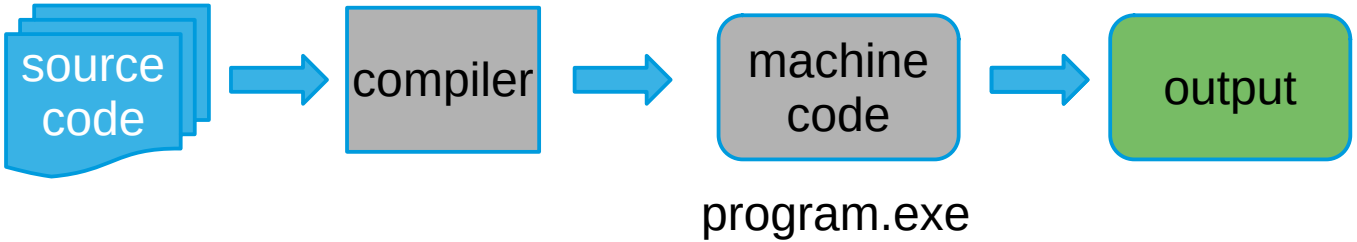
Lecture 1.2: Progamming Basics

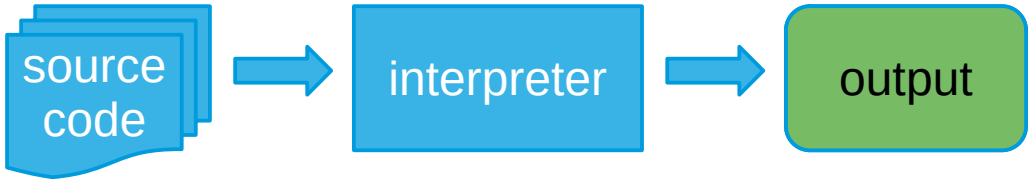
- Source code example:

```python
# Define the numbers
a = 2
b = 3
c = 4

# Add a and b
sum_of_a_and_b = a + b

# Multiply the sum by c
result = sum_of_a_and_b * c

# Print the result in a shorter format
print(f"Result: {result}")
```

Result?

Interpreter vs Compiler





Interpreter vs Compiler

Interpreted Language Workflow: Python

- For Python, which is an interpreted language, you simply run the script/program using the Python interpreter. This means the Python code is executed line by line by the interpreter, without the need for a separate compilation step.

bash Copy code

```
python example.py
```

Interpreter vs Compiler

- ## Compiled Language Workflow

  For C, which is a compiled language, the process involves **two steps:** The source code is first transformed into machine code through compilation, and then the compiled program is run. During compilation pre-compiled parts, so-called libraries are being build into the executable (“linking”).

  1. compilation (creating an “executable”)

```bash
gcc example.c -o example.exe
```

  2. execution (running the “executable”/program)

```bash
./example.exe
```

Libraries

A library in programming is a collection of **pre-compiled** routines, functions, or classes that a program can use to perform specific tasks, thereby avoiding the need to write code from scratch.

- **Code reusability:** Libraries provide a way to reuse code across multiple programs, saving development time and improving efficiency.
- **Abstraction:** Libraries abstract away complex code, making it easier to program complex tasks.
- **Efficiency:** Optimized libraries can improve a program's performance.
- Typical examples: Math or graphics libraries
- **With libraries one can use all the cool stuff other smart people have build on**
- But usage depends on the license model (→ open source software / OSS)

Open Source Software (OSS)

- OSS is code that is designed to be publicly accessible—**anyone** can see, modify, and distribute it.
- Open source software is developed in a decentralized and collaborative way. It is often cheaper, more flexible, and has more longevity than its proprietary peers
- Open source initiative as a movement founded in 1998
- Many open source projects are hosted on GitHub (owned by Microsoft)

https://www.redhat.com/en/topics/open-source/what-is-open-source

Open Source Software: Linux

Linux Operating System as Open-Source Software

- Linux powers a significant portion of the internet's infrastructure. Its the preferred choice for servers and cloud-based applications.
- Accessibility and Community Support: Linux exemplifies the power of community-driven development. This collective effort has resulted in a robust, secure, and highly customizable operating system.
- Driving Innovation in Computing: Linux is at the forefront of innovation in computing. The open-source nature of Linux encourages experimentation and development, speeding up technological advancements and adoption.
- Enabling High-Performance Computing and Scientific Research: Linux is integral to high-performance computing (HPC) and scientific research, sectors where it has become virtually synonymous with supercomputing.

Open Source Software: pytorch for AI

PyTorch as Open-Source Software / Libraries on Advancing Artificial Intelligence

- PyTorch is an open-source machine learning **library** developed by Facebook's AI Research lab (FAIR). It's designed for applications such as **computer vision** and natural language processing
- PyTorch is freely available for anyone to use, modify, and distribute.
- Applications developed with PyTorch include areas such as healthcare (for disease prediction and drug discovery), autonomous vehicles, robotics, and language translation services. These advancements contribute to societal progress and solve complex problems more efficiently.
- Pytorch (and related libraries) was also heavily used for development of **ChatGPT** and similar Large Language Models

Some Vocabulary of Software Development

- Deployment and Infrastructure Management:
  Continuous Integration/Continuous Deployment (CI/CD)

  Source Control

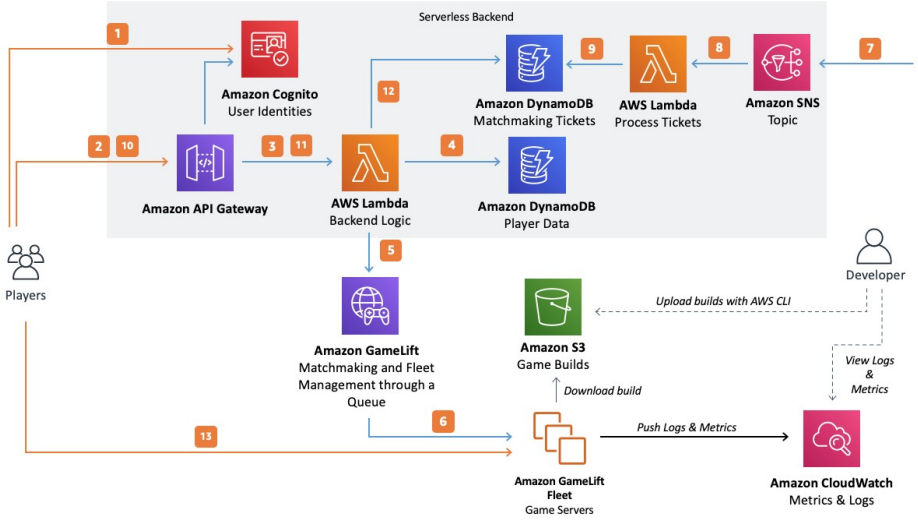
  (Unit) Testing and Quality Assurance (QA)
- User Experience (UX) Design
- Developer Experience

  Integrated Development Environments (IDEs)

  Debugging

Code Complexity

- Apps & infrastructure can get very complicated, in particular for maintenance & updates



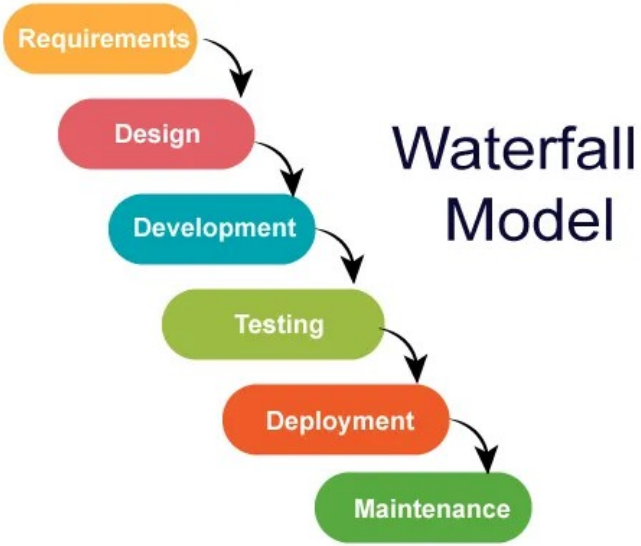
Example:
Game app infrastructure in the cloud

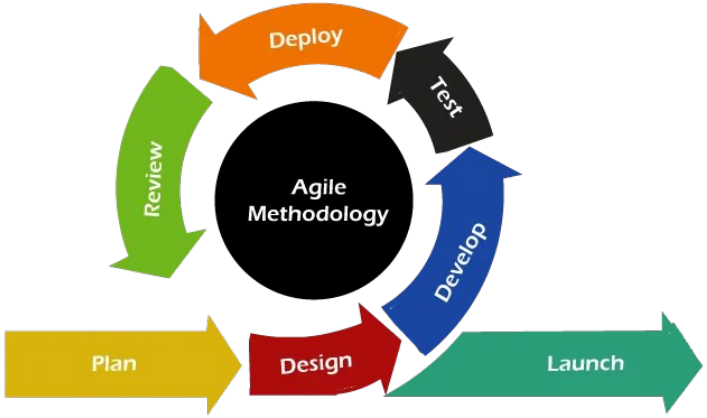
https://aws.amazon.com/blogs/architecture/top-5-featured-architecture-content-for-november/

Software Testing

- **Software needs to be thoroughly tested before deployment**
- Otherwise bugs will annoy the users or do economic or even physical damage
- Unit testing: Testing single functions or small parts of the code, mostly automated
- Integration testing: Testing if multiple components work together as inspected
- Code coverage: How much code (in%) is covered with tests, i.e. which parts of the code have been used in testing

Software Development Life Cycle





https://medium.com/@nayanatharasamarakkody/software-development-life-cycle-sdlc-models-aa18fc085f28

Python Basics

Outlook

- Next Lecture: Data types & structures