Web Applications

Mitsiu Alejandro Carreño Sarabia

Agenda

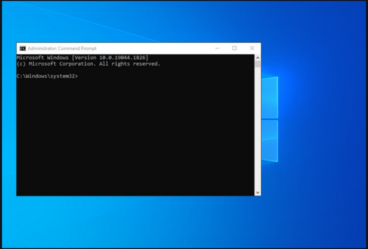
```
├── Class requirements
├── Course structure
├── Math languages
├── Paradigms
└── Homework
```

Class requirements

Terminal mastery

The terminal window is a powerfull tool which will allow us to execute commands and programs, you are required to learn how to use the terminal

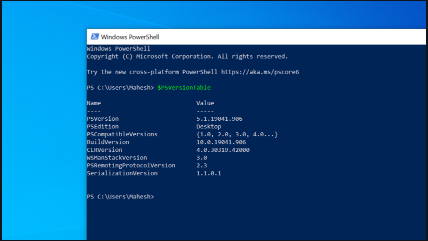
CMD



```
Microsoft Windows [Version 10.0.19044.1826]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32>
```

Powershell



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Mahesh> $PSVersionTable

Name                           Value
----                           -----
PSVersion                      5.1.19041.906
PSEdition                      Desktop
PSCompatibleVersions           {1.0, 2.0, 3.0, 4.0...}
BuildVersion                   10.0.19041.906
CLRVersion                     4.0.30319.42000
WSManStackVersion              3.0
PSRemotingProtocolVersion      2.3
SerializationVersion           1.1.0.1


PS C:\Users\Mahesh>
```

Editor mastery

Most of our time and probably a significant part of your professional time will be spent at your code editor, get to know it!

VsCode

Intellij

NetBeans

Editor mastery

Most of our time and probably a significant part of your professional time will be spent at your code editor, get to know it!

- Search in a single file
- Search in multiple files
- Know filename and file path of open file
- Go to definition
- Split screen
- Go to a specific line in a file
- Find and replace in a single/multiple files

Course structure

Shared slides

All this slides will be shared, as well as source code and extra material so here's my recommendation:

- Take hand written notes in a notebook
- Write down definitions
- Write down examples or analogies which makes sense to you
- Write down your interpretation of the lecture

Course materials

Requirements:

Mandatory:

- Computer
- Master your source code editor
- Commitment
- Computers ready to share screen (adapters)

Optional but recommended:

- Physical notebooks
- Get a lot of tokens
- Disable AI assistants (copilot, chatgpt, etc)
- Disable autocompletion in your editor

Course structure **Partial test**

- 1st Partial = 30%
    - Class exercise / Homework = 10%
    - Theorical evaluation - Functional paradigm = 20%
    - Theorical evaluation - HTML = 20%
    - Practical evaluation - Functional paradigm (paper based, NO computer!) = 25%
    - Practical evaluation - HTML (paper based, NO computer!) = 25%
- 2nd Partial = 30%
    - Class exercise / Homework = 10%
    - Theorical evaluation = 40%
    - Practical evaluation (paper based, NO computer) = 50%
- 3rd Partial = 40%
    - Theorical evaluation = 40%
    - Practical evaluation (paper based, NO computer) = 60%

Course structure

I will giveaway tokens an in-class currency valued as:

1 token = 0.1 extra in your partial grade

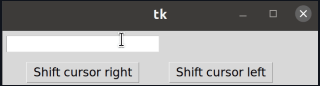
Tokens are earned by showing interest in the class, this broad concept can be implemented as:

- Answering thoughtfull questions
- Asking thoughtfull questions
- Researching class-related topics
- Teaching/expaining to other classmates
- Specific in-class activities
- Showing mastery at activities

My setup

I personally prefer text-based rather than graphical interfaces, it's faster!

- Mis-clicks
- Loose the cursor
- Time spent to move coursor



You can use any editor or environment which suits you, I encourage you to learn your tools so you can set them up for your use and confort.

My setup

But in case of wonder here's a quick overview of my setup:

OS: Fedora/Gnome (Ownership)
https://fedoraproject.org/

Terminal: Kitty (Multiplexer)
https://sw.kovidgoyal.net/kitty/

My setup

But in case of wonder here's a quick overview of my setup:

Editor: Neovim (Fast)
https://neovim.io/

Slides: Presenterm (Slides as code)
https://github.com/mfontanini/presenterm

Wellness

The things you will learn in this class are important, but remember that **it's just a class**. If you're sleeping 4 hours at night, skipping meals or otherwise compromising your health for the sake of this class, **something is wrong**.

Take care of yourself if you are experiencing difficulties, dont hesitate to reach out **talk to me**. I am a **full-time professor and can find my at my cubicule.**

Let's make a plan to help you succeed.

Course intro

- Why this career?
- Do you enjoy writting code?
- Which is the most complex program you've written?
- Which programming languages do you know? Favourite?
- Which editor/tools do you use to write code?
- Do you know what git is?
- Which Operating system do you use?
- Which Operating system did you use on yout OS class last quarter?
- What's your opinion on tech certificates?
- Do you enjoy reading? (Not limited to books)
- To whom do you write code to?
- Did you enjoy your OOP course?

🟩 Math language

What is programming?

Programming is the act of **instructing a computer on how to achieve some kind of operation.**

Programming is **inherently a communicative act.**

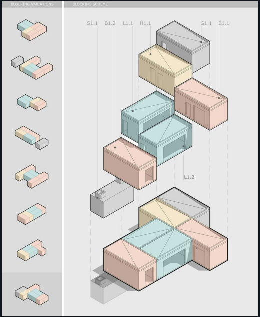
Instructing is the key word. Good communication exists, so what is good programming?

Modularity

Good programming should be **modular**.

Logically distinct parts should be separated, for separate maintenance and reuse.

You should be able to think about a single area of a codebase without needing to concern yourself with unrelated logic.

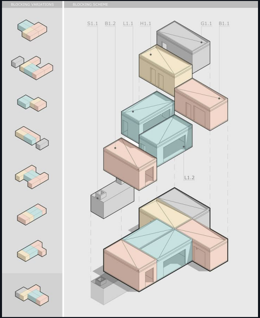


Modularity

Good programming should be **modular**.

Let's suppose we are writting a program to buy items online, why developing the payment processing module, we shouldn't be concerned about authentication!

Programs grow in complexity at an impressive rate, our memory can't keep up.



Maintainability

Good programming should be **maintainable.**

Programs should be written with future maintainability in mind.

Code written expressively, and has future expansion in mind, will be easiest to maintain.

We write code for our future selfs, for our development team and lastly for the computer.

Math language (Values and expressions)

> Expression: Combination of one or more constants, variables, functions, and operators

$$5 + 2$$

Expressions can often be **simplified or evaluated**.

$$5 + 2 \Rightarrow 7$$

> Value: The result of a calculation (a **final answer** that cannot be simplified further)

Math language (Values and expressions)

We use the

$$\Longrightarrow$$

simbol to denote stepping (or reducing) of expressions, which means to simplify an expression by one step.

Math language (Values and expressions)

$$(5+2)*3$$

Let's evaluate this expression:

$$(5+2)*3 \Rightarrow 7*3$$
$$\Rightarrow 21$$

> Expression: Combination of one or more constants, variables, functions, and operators

> Value: The result of a calculation (a **final answer** that cannot be simplified further)

Math	Expression	Value
$(5+2)*3$	✅	❌
$7*3$	✅	❌
21	✅	✅

Values are expressions!

Math language (Values and expressions)

Values are expressions!

It's a mathematical phrase, just a very simple one.

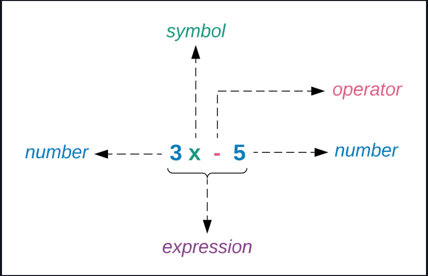
$$21$$

$$3$$

$$-4$$

Math language (Values and expressions)

Let's analyse an expression a little bit harder:



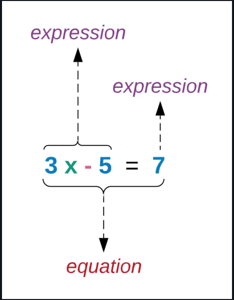
Since **x** is a variable, it's value will also change depending on the context of a problem.

What's the value if **x = 2**?

Also we can put a constraint on **x**

Math language (Values and expressions)

Also we can put a constraint on **x**



The expression on the left must evaluate to 7.

What are the possible values **x** can assume?

Math language (Values and expressions)

What happens if we try to evaluate:

$$(5+2)*3x = (4+3)*x$$

$$\begin{aligned}(5+2)*3x = (4+3)*x &\Rightarrow 7*3x = 7*x \\ &\Rightarrow 21x = 7x \\ &\Rightarrow 21x - 7x = 0 \\ &\Rightarrow 14x = 0 \\ &\Rightarrow x = 0\end{aligned}$$

Math language (Values and expressions)

Math gives us this clear and concise language, with clear rules and simbols.

$$\begin{aligned}(5+2)*3 &\Rightarrow 7*3 \\ &\Rightarrow 21\end{aligned}$$

$$\begin{aligned}(5+2)*3x=(4+3)*x &\Rightarrow 7*3x=7*x \\ &\Rightarrow 21x=7x \\ &\Rightarrow 21x-7x=0 \\ &\Rightarrow 14x=0 \\ &\Rightarrow x=0\end{aligned}$$

Math language (Values and expressions)

Finally let's try to evaluate:

$$x = x + 1$$

$$\begin{aligned} x = x + 1 &\Rightarrow x - x = 1 \\ &\Rightarrow 0 = 1 \\ &\Rightarrow \text{🤯} \end{aligned}$$

This equation is a **contradiction**

Paradigms

A tale of two equals

$$x = x + 1$$

In math equals means equality the left expression has to evaluate to the same value that the right expression.

Yet we use it often while programming, for example check this java code:

```java
int x = 0;
for (int i = 0; i < 5; i++){
    x = x + 1;  <------
}
```

In imperative programs equals means an assignment; x is now equal to the previous value of x and we have added to it.

Double equals does check for equality which evaluates to true or false.

Imperative programming

Let's revisit our Java snippet

```java
int x = 0;
for (int i = 0; i < 5; i++){
    x = x + 1;
}
```

When we instruct **x = x + 1;** we are changing the state of our program.
x has a new value, we change the world, from one where x was 0, to be one where x is 1.

Imperative programming

```
1 int x = 0;
2 for (int i = 0; i < 5; i++){
3     x = x + 1;
4 }
```

Imagine we are tracking a bug that occurs in line 3, this is problematic because, understanding a programs entails not only understanding what the code does, but knowing the entire history of the program up until that point.

Imperative programming

Let's look at other example of non-functional code in Java:

```java
class Inc {
  static int count = 0;

  static int increment(){
    count +=1;

    return count;
  }
}
```

If i call Inc.increment() what does it return?

The answer: **it depends**. On the first call, it returns 1, on the second call, 2 and so on.

State

Previously we mentioned:

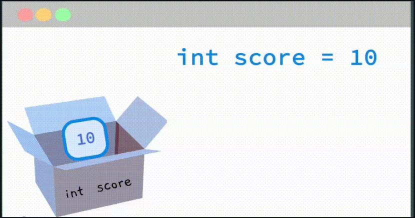
```
int x = 0;
for (int i = 0; i < 5; i++){
    x = x + 1;
}
```

When we instruct **x = x + 1;** we are changing the state of our program. x has a new value, we change the world, from one where x was 0, to be one where x is 1.

A computer program stores data in variables, which represent storage locations in the computers memory. The contents of these memory locations, at any given point in the program execution, are called the program state.

State

Usually state is represented as:



Credit: Transcode
https://www.youtube.com/watch?v=ghCbURMWBD8

This approach lead to several interesting situations:

- A box might be empty
- The box contents can change at any time

State

This behaviour can be found in the real world:

1. Search for me in my office

    - I might or not be at my office
    - My office has certain schedule that changes every academic cycle

2. Want to use a univerity bathroom stall

    - The bathroom might be occupied or not
    - The bathroom might be closed by the cleaning services
    - The bathroom sink might have no water

The state is unpredictable because there are several processes that modify its state.

State

It's such a hassle we always have to verify the state, make sure we can proceed, wouldn't it be great if we have a single bathroom stall (or teacher) exclusive for us?

Which state is always available for us.

(Recap) A new paradigm

Programming paradigm

1. A high level model to conceptualize and structure a computer program implementation

Which programming paradigms do you know?

Which rules and structures does object oriented paradigm enforce?

Functional paradigm

In this course we will be learning **the functional paradigm** where we avoid modification of state. Functional programming is **an improvement on our ability to communicate as programmers.**

Programming, Imperatively

Computation by **modifying the computer's state**

```
x = 2;
x + x
```

Programming, Functionally

Computation by **reduction of expressons to values**

```
2 + 2
```

State

In statefull programs, we use commands like **x = 2** to change the word, to be one where x + x is 4. This doesn't stop another part ot the program to change the value of x and re-compute the result:

```
x = 2;
x = 5;
x + x
```

In functional programs, we apply simplifying rules to expressions like 2 + 2, to obtain the value of 4

These expressions are **disjoint**, in that evaluation of one expression is unrelated to the evaluation of another.

Last word

Functional programming allows reasoning about programs and their subcomponents in the same way that you would reason about a mathematical expression.

We’re not just in the business of writing code, but correct code!

Homework

Homework

1. **Bring your computer next session**

2. **Master your terminal:**

   - `cd <path>` Change to a specific directory
   - `cd ..` Go to parent directory
   - `pwd` Print current directory

3. **Master your code editor:**

   - Search in a single file
   - Search in multiple files
   - Know filename and file path of open file
   - Go to definition
   - Split screen
   - Go to a specific line in a file
   - Find and replace in a single/multiple files

Homework

4. Master your keyword

- How to keypress () [] {}
- https://monkeytype.com/
- Practice PascalCase with shift key

5. Upskill your english