COMP110: Principles of Computing

2: Algorithms

Worksheet 2

Available now!

FALMOUTH
UNIVERSITY

Programming languages and paradigms

What is a programming language?

What is a programming language?

- A **program** is a sequence of instructions for a computer to perform a specific task

What is a programming language?

- A **program** is a sequence of instructions for a computer to perform a specific task
- A **programming language** is a formal language for communicating these sequences of instructions

Which is the best programming language?

Which is the best programming language?

- There is no “best” programming language

Which is the best programming language?

- There is no “best” programming language
- There are hundreds of programming languages, each better suited to some tasks than others

Which is the best programming language?

- There is no “best” programming language
- There are hundreds of programming languages, each better suited to some tasks than others
- Sometimes your choice is dictated by your choice of platform, framework, game engine etc.

Which is the best programming language?

- There is no "best" programming language
- There are hundreds of programming languages, each better suited to some tasks than others
- Sometimes your choice is dictated by your choice of platform, framework, game engine etc.
- To become a better programmer (and maximise your employability) you should learn several languages (but one at a time!)

Low vs high level

Low vs high level

- **Low level languages** give the programmer direct control over the hardware

Low vs high level

- **Low level languages** give the programmer direct control over the hardware
- **High level languages** give the programmer **abstraction**, hiding the details of the hardware

Low vs high level

- **Low level languages** give the programmer direct control over the hardware
- **High level languages** give the programmer **abstraction**, hiding the details of the hardware
- High level languages trade efficiency for ease of programming

Low vs high level

- **Low level languages** give the programmer direct control over the hardware
- **High level languages** give the programmer **abstraction**, hiding the details of the hardware
- High level languages trade efficiency for ease of programming
- Lower level languages were once the choice of game programmers, but advances in hardware mean that higher level languages are often a better choice

Programming paradigms

Programming paradigms

- **Imperative**: program is a simple sequence of instructions, with **goto** instructions for program flow

Programming paradigms

- **Imperative**: program is a simple sequence of instructions, with **goto** instructions for program flow
- **Structured**: like imperative, but with **control structures** (loops, conditionals etc.)

Programming paradigms

- **Imperative**: program is a simple sequence of instructions, with **goto** instructions for program flow
- **Structured**: like imperative, but with **control structures** (loops, conditionals etc.)
- **Procedural**: structured program is broken down into **procedures**

Programming paradigms

- **Imperative**: program is a simple sequence of instructions, with **goto** instructions for program flow
- **Structured**: like imperative, but with **control structures** (loops, conditionals etc.)
- **Procedural**: structured program is broken down into **procedures**
- **Object-oriented**: related procedures and data are grouped into **objects**

Programming paradigms

- **Imperative**: program is a simple sequence of instructions, with **goto** instructions for program flow
- **Structured**: like imperative, but with **control structures** (loops, conditionals etc.)
- **Procedural**: structured program is broken down into **procedures**
- **Object-oriented**: related procedures and data are grouped into **objects**
- **Functional**: procedures are treated as mathematical objects that can be passed around and manipulated

Programming paradigms

- **Imperative**: program is a simple sequence of instructions, with **goto** instructions for program flow
- **Structured**: like imperative, but with **control structures** (loops, conditionals etc.)
- **Procedural**: structured program is broken down into **procedures**
- **Object-oriented**: related procedures and data are grouped into **objects**
- **Functional**: procedures are treated as mathematical objects that can be passed around and manipulated
- **Declarative**: does not define the control flow of a program, but rather defines logical relations

Which paradigm?

Which paradigm?

- **Imperative** and **structured** languages are mainly of historical interest

Which paradigm?

- **Imperative** and **structured** languages are mainly of historical interest
- Most commonly used languages today are a mixture of **procedural** and **object-oriented** paradigms, with many also incorporating ideas from **functional** programming

Which paradigm?

- **Imperative** and **structured** languages are mainly of historical interest
- Most commonly used languages today are a mixture of **procedural** and **object-oriented** paradigms, with many also incorporating ideas from **functional** programming
- Purely **functional** languages (e.g. Haskell, F#) are mainly used in academia, but favoured by some programmers

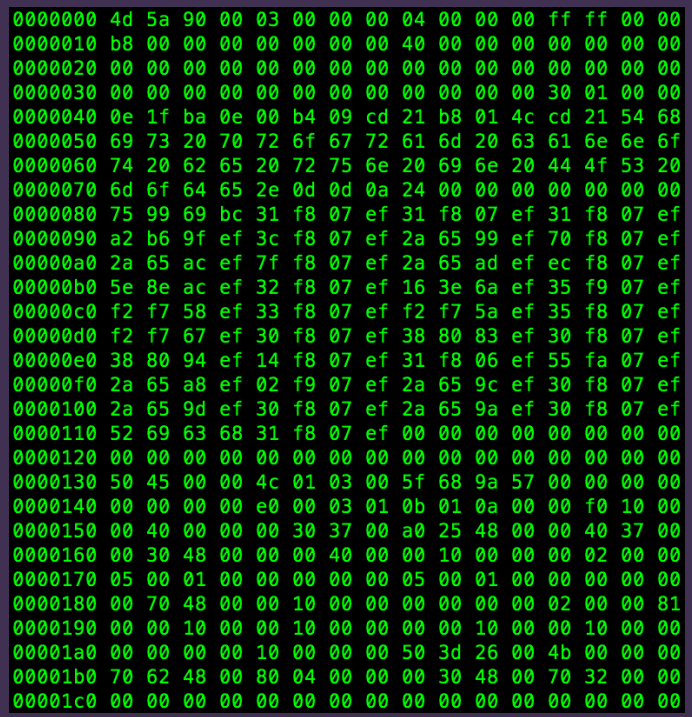
Which paradigm?

- **Imperative** and **structured** languages are mainly of historical interest
- Most commonly used languages today are a mixture of **procedural** and **object-oriented** paradigms, with many also incorporating ideas from **functional** programming
- Purely **functional** languages (e.g. Haskell, F#) are mainly used in academia, but favoured by some programmers
- Purely **declarative** languages have uses in academia and some special-purpose languages

Machine code

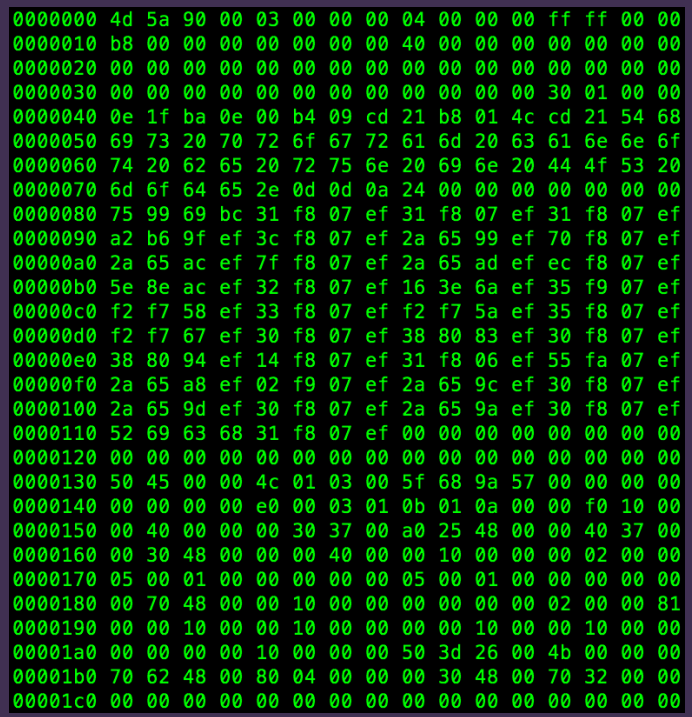
```
0000000 4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00
0000010 b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00
0000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000030 00 00 00 00 00 00 00 00 00 00 00 00 30 01 00 00
0000040 0e 1f ba 0e 00 b4 09 cd 21 b8 01 4c cd 21 54 68
0000050 69 73 20 70 72 6f 67 72 61 6d 20 63 61 6e 6e 6f
0000060 74 20 62 65 20 72 75 6e 20 69 6e 20 44 4f 53 20
0000070 6d 6f 64 65 2e 0d 0d 0a 24 00 00 00 00 00 00 00
0000080 75 99 69 bc 31 f8 07 ef 31 f8 07 ef 31 f8 07 ef
0000090 a2 b6 9f ef 3c f8 07 ef 2a 65 99 ef 70 f8 07 ef
00000a0 2a 65 ac ef 7f f8 07 ef 2a 65 ad ef ec f8 07 ef
00000b0 5e 8e ac ef 32 f8 07 ef 16 3e 6a ef 35 f9 07 ef
00000c0 f2 f7 58 ef 33 f8 07 ef f2 f7 5a ef 35 f8 07 ef
00000d0 f2 f7 67 ef 30 f8 07 ef 38 80 83 ef 30 f8 07 ef
00000e0 38 80 94 ef 14 f8 07 ef 31 f8 06 ef 55 fa 07 ef
00000f0 2a 65 a8 ef 02 f9 07 ef 2a 65 9c ef 30 f8 07 ef
0000100 2a 65 9d ef 30 f8 07 ef 2a 65 9a ef 30 f8 07 ef
0000110 52 69 63 68 31 f8 07 ef 00 00 00 00 00 00 00 00
0000120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000130 50 45 00 00 4c 01 03 00 5f 68 9a 57 00 00 00 00
0000140 00 00 00 00 e0 00 03 01 0b 01 0a 00 00 f0 10 00
0000150 00 40 00 00 00 30 37 00 a0 25 48 00 00 40 37 00
0000160 00 30 48 00 00 00 40 00 00 10 00 00 00 02 00 00
0000170 05 00 01 00 00 00 00 00 05 00 01 00 00 00 00 00
0000180 00 70 48 00 00 10 00 00 00 00 00 00 02 00 00 81
0000190 00 00 10 00 00 10 00 00 00 00 10 00 00 10 00 00
00001a0 00 00 00 00 00 10 00 00 00 50 3d 26 00 4b 00 00 00
00001b0 70 62 48 00 80 04 00 00 00 30 48 00 70 32 00 00
00001c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Machine code



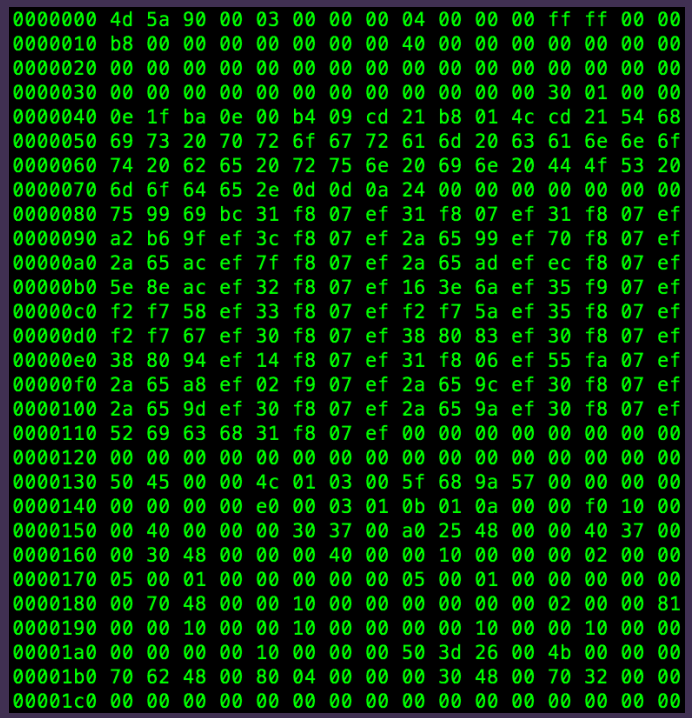
- Programs are represented as sequences of **numbers** specifying **machine instructions**

Machine code



- Programs are represented as sequences of **numbers** specifying **machine instructions**
- More on this later in the module

Machine code



- Programs are represented as sequences of **numbers** specifying **machine instructions**
- More on this later in the module
- Nobody has actually written programs in machine code since the 1960s...

Assembly language

```
section     .text
global      _start

_start:

    mov     edx,len
    mov     ecx,msg
    mov     ebx,1
    mov     eax,4
    int     0x80

    mov     eax,1
    int     0x80

section     .data

msg     db  'Hello, world!',0xa
len     equ $ - msg
```

Assembly language

```
section     .text
global      _start

_start:

    mov     edx,len
    mov     ecx,msg
    mov     ebx,1
    mov     eax,4
    int     0x80

    mov     eax,1
    int     0x80

section     .data

msg     db  'Hello, world!',0xa
len     equ $ - msg
```

- Each line of assembly code translates **directly** to an instruction of machine code

Assembly language

```
section     .text
global      _start

_start:

    mov     edx,len
    mov     ecx,msg
    mov     ebx,1
    mov     eax,4
    int     0x80

    mov     eax,1
    int     0x80

section     .data

msg     db  'Hello, world!',0xa
len     equ $ - msg
```

- Each line of assembly code translates **directly** to an instruction of machine code
- Commonly used for games in the 70s/80s/90s, but hardly ever used now

Assembly language

```
section     .text
global      _start

_start:

    mov     edx,len
    mov     ecx,msg
    mov     ebx,1
    mov     eax,4
    int     0x80

    mov     eax,1
    int     0x80

section     .data

msg     db  'Hello, world!',0xa
len     equ $ - msg
```

- Each line of assembly code translates **directly** to an instruction of machine code
- Commonly used for games in the 70s/80s/90s, but hardly ever used now
- Allows very fine control over the hardware...

Assembly language

```
section     .text
global      _start

_start:

    mov     edx,len
    mov     ecx,msg
    mov     ebx,1
    mov     eax,4
    int     0x80

    mov     eax,1
    int     0x80

section     .data

msg     db  'Hello, world!',0xa
len     equ $ - msg
```

- Each line of assembly code translates **directly** to an instruction of machine code
- Commonly used for games in the 70s/80s/90s, but hardly ever used now
- Allows very fine control over the hardware...
- ... but difficult to use as there is no **abstraction**

Assembly language

```
section     .text
global      _start

_start:

    mov     edx,len
    mov     ecx,msg
    mov     ebx,1
    mov     eax,4
    int     0x80

    mov     eax,1
    int     0x80

section     .data

msg     db  'Hello, world!',0xa
len     equ $ - msg
```

- Each line of assembly code translates **directly** to an instruction of machine code
- Commonly used for games in the 70s/80s/90s, but hardly ever used now
- Allows very fine control over the hardware...
- ... but difficult to use as there is no **abstraction**
- Also not portable between CPU architectures

C++

```cpp
#include "stdafx.h"
#include "GameObject.h"
#include "CoinGame.h"

GameObject::GameObject(CoinGame* game, Texture* sprite)
	: game(game), sprite(sprite), isDead(false)
{
	x = rand() % CoinGame::WINDOW_WIDTH;
	y = rand() % CoinGame::WINDOW_HEIGHT;
}


GameObject::~GameObject()
{
}

void GameObject::render(SDL_Renderer* renderer)
{
	sprite->render(renderer, x, y, CoinGame::SPRITE_SIZE, CoinGame::SPRITE_SIZE);
}

bool GameObject::checkCollision(int otherX, int otherY)
{
	double distance = sqrt(pow(otherX - x, 2) + pow(otherY - y, 2));
	return (distance < CoinGame::SPRITE_SIZE / 2);
}
```

C++

- Initially an object-oriented extension for the procedural language C

```
#include "stdafx.h"
#include "GameObject.h"
#include "CoinGame.h"

GameObject::GameObject(CoinGame* game, Texture* sprite)
        : game(game), sprite(sprite), isDead(false)
{
        x = rand() % CoinGame::WINDOW_WIDTH;
        y = rand() % CoinGame::WINDOW_HEIGHT;
}


GameObject::~GameObject()
{
}

void GameObject::render(SDL_Renderer* renderer)
{
        sprite->render(renderer, x, y, CoinGame::SPRITE_SIZE, CoinGame::SPRITE_SIZE);
}

bool GameObject::checkCollision(int otherX, int otherY)
{
        double distance = sqrt(pow(otherX - x, 2) + pow(otherY - y, 2));
        return (distance < CoinGame::SPRITE_SIZE / 2);
}
```

C++

- Initially an object-oriented extension for the procedural language C
- Low level (though higher level than assembly)

```
#include "stdafx.h"
#include "GameObject.h"
#include "CoinGame.h"

GameObject::GameObject(CoinGame* game, Texture* sprite)
        : game(game), sprite(sprite), isDead(false)
{
        x = rand() % CoinGame::WINDOW_WIDTH;
        y = rand() % CoinGame::WINDOW_HEIGHT;
}


GameObject::~GameObject()
{
}

void GameObject::render(SDL_Renderer* renderer)
{
        sprite->render(renderer, x, y, CoinGame::SPRITE_SIZE, CoinGame::SPRITE_SIZE);
}

bool GameObject::checkCollision(int otherX, int otherY)
{
        double distance = sqrt(pow(otherX - x, 2) + pow(otherY - y, 2));
        return (distance < CoinGame::SPRITE_SIZE / 2);
}
```

C++

- Initially an object-oriented extension for the procedural language C
- Low level (though higher level than assembly)
- Used by developers of game engines, and games using many popular “AAA” engines (Unreal, Source, CryEngine, ...)

```
#include "stdafx.h"
#include "GameObject.h"
#include "CoinGame.h"

GameObject::GameObject(CoinGame* game, Texture* sprite)
        : game(game), sprite(sprite), isDead(false)
{
        x = rand() % CoinGame::WINDOW_WIDTH;
        y = rand() % CoinGame::WINDOW_HEIGHT;
}


GameObject::~GameObject()
{
}

void GameObject::render(SDL_Renderer* renderer)
{
        sprite->render(renderer, x, y, CoinGame::SPRITE_SIZE, CoinGame::SPRITE_SIZE);
}

bool GameObject::checkCollision(int otherX, int otherY)
{
        double distance = sqrt(pow(otherX - x, 2) + pow(otherY - y, 2));
        return (distance < CoinGame::SPRITE_SIZE / 2);
}
```

C++

```
#include "stdafx.h"
#include "GameObject.h"
#include "CoinGame.h"

GameObject::GameObject(CoinGame* game, Texture* sprite)
        : game(game), sprite(sprite), isDead(false)
{
        x = rand() % CoinGame::WINDOW_WIDTH;
        y = rand() % CoinGame::WINDOW_HEIGHT;
}


GameObject::~GameObject()
{
}

void GameObject::render(SDL_Renderer* renderer)
{
        sprite->render(renderer, x, y, CoinGame::SPRITE_SIZE, CoinGame::SPRITE_SIZE);
}

bool GameObject::checkCollision(int otherX, int otherY)
{
        double distance = sqrt(pow(otherX - x, 2) + pow(otherY - y, 2));
        return (distance < CoinGame::SPRITE_SIZE / 2);
}
```

- Initially an object-oriented extension for the procedural language C
- Low level (though higher level than assembly)
- Used by developers of game engines, and games using many popular “AAA” engines (Unreal, Source, CryEngine, ...)
- Also used by developers of operating systems and embedded systems, but falling out of favour with other software developers

High level languages in games

High level languages in games

Often favoured by smaller indie teams for rapid development

High level languages in games

Often favoured by smaller indie teams for rapid development

- C# (Unity)

High level languages in games

Often favoured by smaller indie teams for rapid development

- C# (Unity)
- Python (EVE Online, Pygame, Ren'py)

High level languages in games

Often favoured by smaller indie teams for rapid development

- C# (Unity)
- Python (EVE Online, Pygame, Ren'py)
- JavaScript/TypeScript (HTML5 browser games, Electron, node.js)

High level languages in games

Often favoured by smaller indie teams for rapid development

- C# (Unity)
- Python (EVE Online, Pygame, Ren'py)
- JavaScript/TypeScript (HTML5 browser games, Electron, node.js)
- Objective-C, Swift (iOS games)

High level languages in games

Often favoured by smaller indie teams for rapid development

- C# (Unity)
- Python (EVE Online, Pygame, Ren'py)
- JavaScript/TypeScript (HTML5 browser games, Electron, node.js)
- Objective-C, Swift (iOS games)
- Java (Minecraft, Android games)

High level languages in games

Often favoured by smaller indie teams for rapid development

- C# (Unity)
- Python (EVE Online, Pygame, Ren'py)
- JavaScript/TypeScript (HTML5 browser games, Electron, node.js)
- Objective-C, Swift (iOS games)
- Java (Minecraft, Android games)

There are many others, but these are the most commonly used in game development

High level languages in other domains

High level languages in other domains

- Data science: Python, R, Matlab, ...

High level languages in other domains

- Data science: Python, R, Matlab, ...
- Web: JavaScript, TypeScript, Python, Ruby, PHP, ...

High level languages in other domains

- Data science: Python, R, Matlab, ...
- Web: JavaScript, TypeScript, Python, Ruby, PHP, ...
- Hardware: VHDL, MicroPython, ...

High level languages in other domains

- Data science: Python, R, Matlab, ...
- Web: JavaScript, TypeScript, Python, Ruby, PHP, ...
- Hardware: VHDL, MicroPython, ...
- Creative computing: Processing, SuperCollider, Sonic Pi, ...

Scripting languages in games

Scripting languages in games

Many games use scripting languages in addition to their main development language

Scripting languages in games

Many games use scripting languages in addition to their main development language

- Lua (many AAA games)

Scripting languages in games

Many games use scripting languages in addition to their main development language

- Lua (many AAA games)
- Bespoke languages (many AAA games)

Scripting languages in games

Many games use scripting languages in addition to their main development language

- Lua (many AAA games)
- Bespoke languages (many AAA games)

Some game engines have their own scripting language

Scripting languages in games

Many games use scripting languages in addition to their main development language

- Lua (many AAA games)
- Bespoke languages (many AAA games)

Some game engines have their own scripting language

- Blueprint (Unreal Engine)

Scripting languages in games

Many games use scripting languages in addition to their main development language

- Lua (many AAA games)
- Bespoke languages (many AAA games)

Some game engines have their own scripting language

- Blueprint (Unreal Engine)
- GDScript (Godot)

Scripting languages in games

Many games use scripting languages in addition to their main development language

- Lua (many AAA games)
- Bespoke languages (many AAA games)

Some game engines have their own scripting language

- Blueprint (Unreal Engine)
- GDScript (Godot)
- GML (GameMaker)

Scripting languages

Scripting languages

Outside of games, "scripting" can refer to command-line scripting

Scripting languages

Outside of games, "scripting" can refer to command-line scripting

- Bash scripts (Unix)

Scripting languages

Outside of games, "scripting" can refer to command-line scripting

- Bash scripts (Unix)
- Batch files, PowerShell (Windows)

Scripting languages

Outside of games, "scripting" can refer to command-line scripting

- Bash scripts (Unix)
- Batch files, PowerShell (Windows)

Some languages blur the line between scripting and programming, depending on usage

Scripting languages

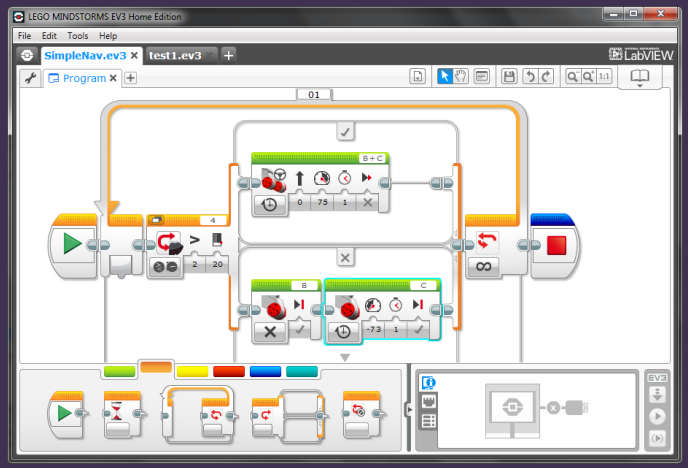
Outside of games, "scripting" can refer to command-line scripting

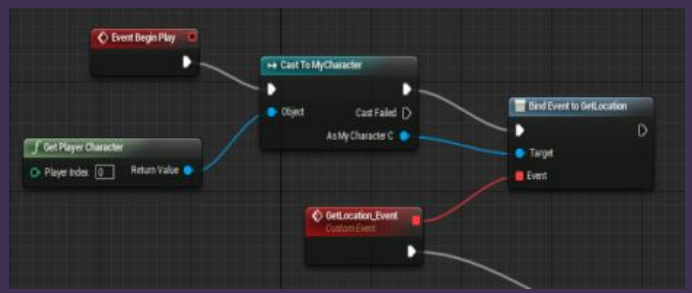
- Bash scripts (Unix)
- Batch files, PowerShell (Windows)

Some languages blur the line between scripting and programming, depending on usage

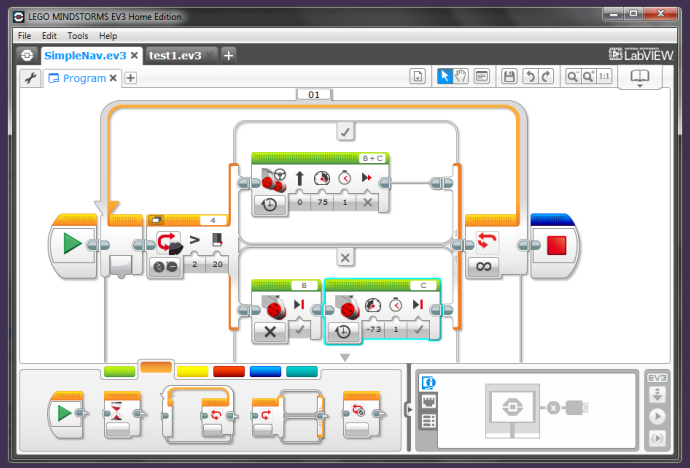
- Python, Ruby, Perl, Lua...

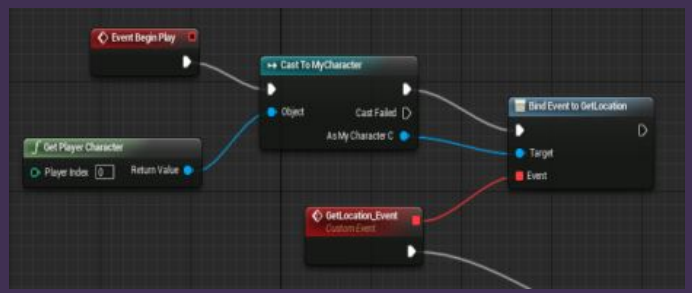
Visual programming languages





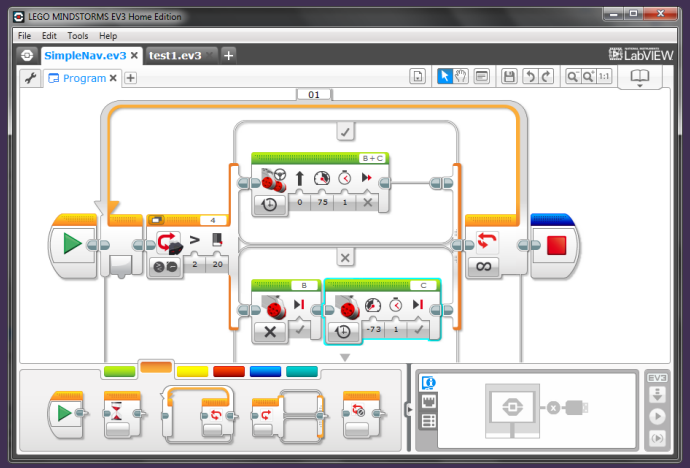
Visual programming languages

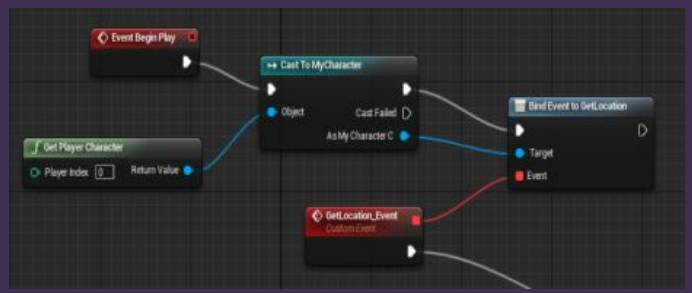




Based on connecting graphical blocks rather than writing code as text

Visual programming languages

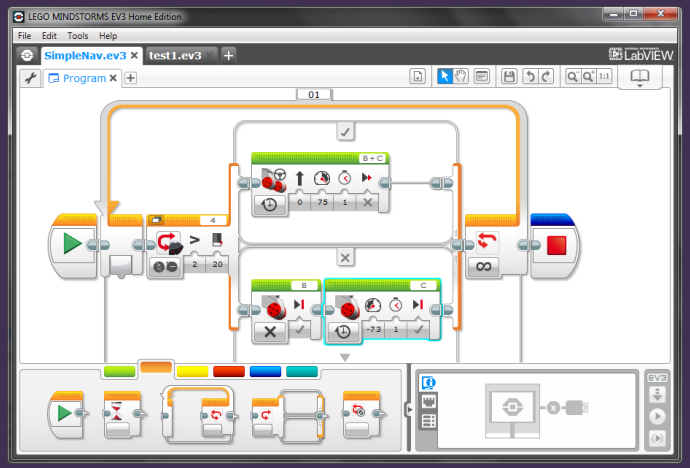


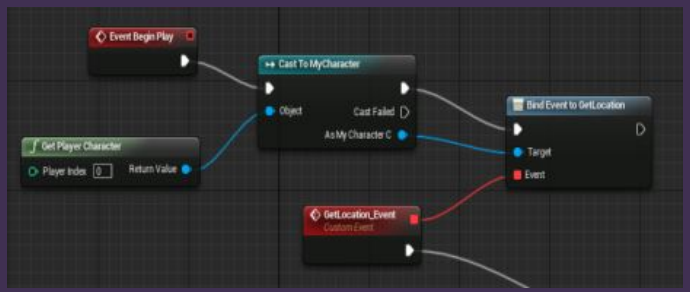


Based on connecting graphical blocks rather than writing code as text

- Scratch

Visual programming languages

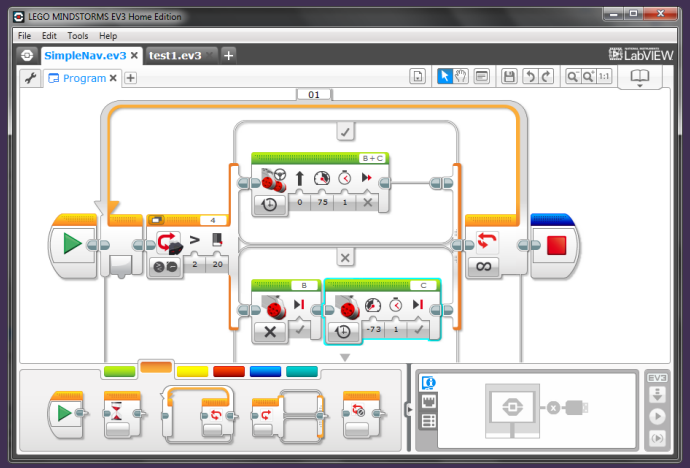


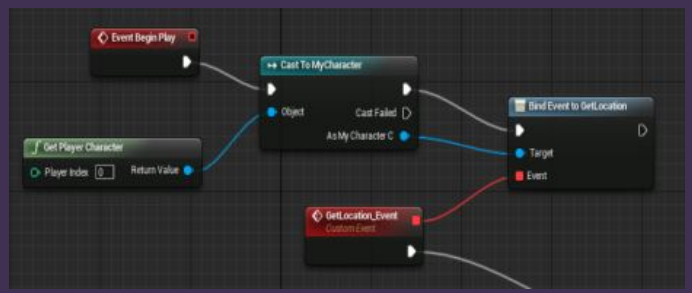


Based on connecting graphical blocks rather than writing code as text

- Scratch
- Lego Mindstorms

Visual programming languages

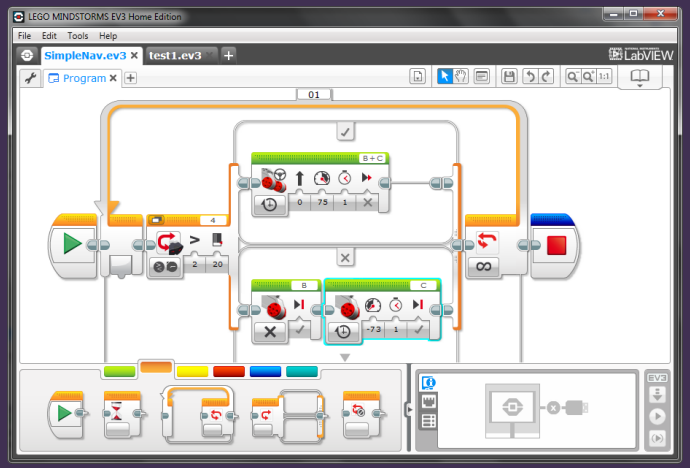


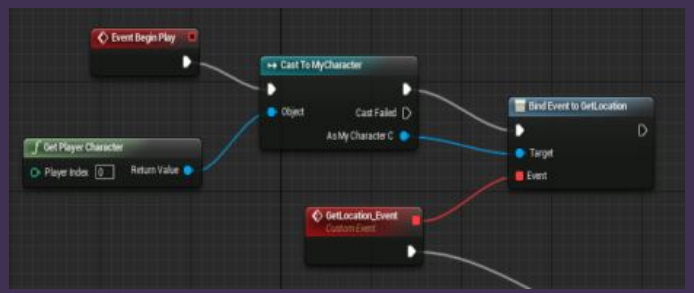


Based on connecting graphical blocks rather than writing code as text

- Scratch
- Lego Mindstorms
- Blueprint (Unreal)

Visual programming languages





Based on connecting graphical blocks rather than writing code as text

- Scratch
- Lego Mindstorms
- Blueprint (Unreal)

Note: despite the name, Microsoft Visual Studio is **not** a visual programming environment!

Special purpose languages

Special purpose languages

- SQL (database queries)

Special purpose languages

- SQL (database queries)
- GLSL, HLSL (GPU shader programs)

Special purpose languages

- SQL (database queries)
- GLSL, HLSL (GPU shader programs)
- LEX, YACC (script interpreters)

Markup languages

Markup languages

Not to be confused with programming languages...

Markup languages

Not to be confused with programming languages...

- HTML, CSS (web pages)

Markup languages

Not to be confused with programming languages...

- HTML, CSS (web pages)
- LaTeX, Markdown (documentation)

Markup languages

Not to be confused with programming languages...

- HTML, CSS (web pages)
- LaTeX, Markdown (documentation)
- XML, JSON (data storage)

Which programming language is most popular?

`http://githut.info`

“Family tree” of programming languages

https://www.levenez.com/lang/lang.pdf

FALMOUTH
UNIVERSITY

Algorithms

What is an algorithm?

What is an algorithm?

A **sequence of instructions** which can be followed **step by step** to perform a **(computational) task**.

Algorithms historically

Algorithms historically

- Named after Muhammad ibn Musa al-Khwarizmi (c. 780–850), Persian mathematician

Algorithms historically

- Named after Muhammad ibn Musa al-Khwarizmi (c. 780–850), Persian mathematician
- Used in mathematics to describe steps for calculations

Algorithms historically

- Named after Muhammad ibn Musa al-Khwarizmi (c. 780–850), Persian mathematician
- Used in mathematics to describe steps for calculations
  - E.g. Euclid's algorithm for finding the greatest common divisor of two numbers

Algorithms historically

- Named after Muhammad ibn Musa al-Khwarizmi (c. 780–850), Persian mathematician
- Used in mathematics to describe steps for calculations
  - E.g. Euclid's algorithm for finding the greatest common divisor of two numbers
- Computers developed as machines for carrying out mathematical algorithms

Programs vs algorithms

Programs vs algorithms

- A program is **specific** to a particular programming language and/or machine

Programs vs algorithms

- A program is **specific** to a particular programming language and/or machine
- An algorithm is **general**

Programs vs algorithms

- A program is **specific** to a particular programming language and/or machine
- An algorithm is **general**
- An algorithm must be **implemented** as a program before a computer can run it

Programs vs algorithms

- A program is **specific** to a particular programming language and/or machine
- An algorithm is **general**
- An algorithm must be **implemented** as a program before a computer can run it
- An algorithm generally performs **one task**, whereas a program may perform **many**

Programs vs algorithms

- A program is **specific** to a particular programming language and/or machine
- An algorithm is **general**
- An algorithm must be **implemented** as a program before a computer can run it
- An algorithm generally performs **one task**, whereas a program may perform **many**
  - E.g. Microsoft Word is not an algorithm, but it implements many algorithms

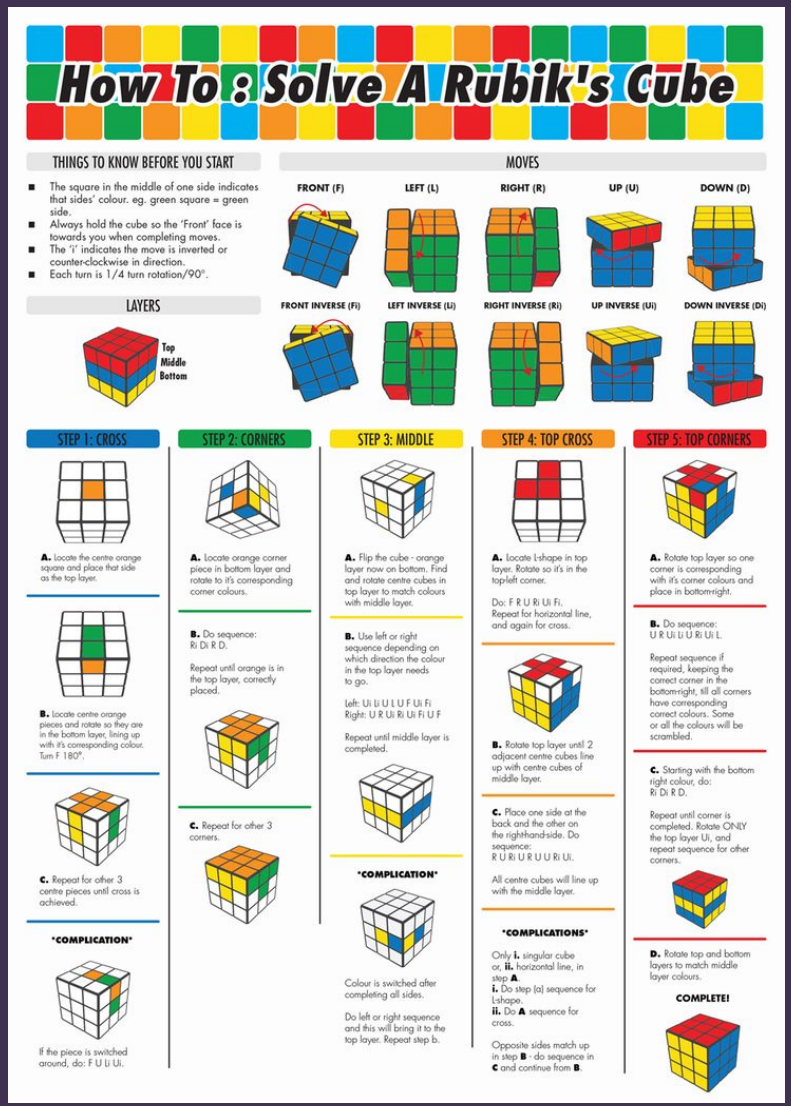
Programs vs algorithms

- A program is **specific** to a particular programming language and/or machine
- An algorithm is **general**
- An algorithm must be **implemented** as a program before a computer can run it
- An algorithm generally performs **one task**, whereas a program may perform **many**
  - E.g. Microsoft Word is not an algorithm, but it implements many algorithms
  - E.g. it implements an algorithm for determining where to break a line of text, how much space to add to centre a line, etc.

Algorithms outside computing

1. Preheat the oven to 180C, gas 4.
2. Beat together the eggs, flour, caster sugar, butter and baking powder until smooth in a large mixing bowl.
3. Put the cocoa in separate mixing bowl, and add the water a little at a time to make a stiff paste. Add to the cake mixture.
4. Turn into the prepared tins, level the top and bake in the preheated oven for about 20-25 mins, or until shrinking away from the sides of the tin and springy to the touch.
5. Leave to cool in the tin, then turn on to a wire rack to become completely cold before icing.
6. To make the icing: measure the cream and chocolate into a bowl and carefully melt over a pan of hot water over a low heat, or gently in the microwave for 1 min (600w microwave). Stir until melted, then set aside to cool a little and to thicken up.
7. To ice the cake: spread the apricot jam on the top of each cake. Spread half of the ganache icing on the top of the jam on one of the cakes, then lay the other cake on top, sandwiching them together.
8. Use the remaining ganache icing to ice the top of the cake in a swirl pattern. Dust with icing sugar to serve.

Algorithms outside computing



Why algorithms?

Why algorithms?

- Allow for common computations to have **common solutions**

Why algorithms?

- Allow for common computations to have **common solutions**
- **Algorithm strategies** give widely applicable approaches for solving problems

Why algorithms?

- Allow for common computations to have **common solutions**
- **Algorithm strategies** give widely applicable approaches for solving problems
- Can **prove** mathematically that an algorithm does what it is supposed to

Why algorithms?

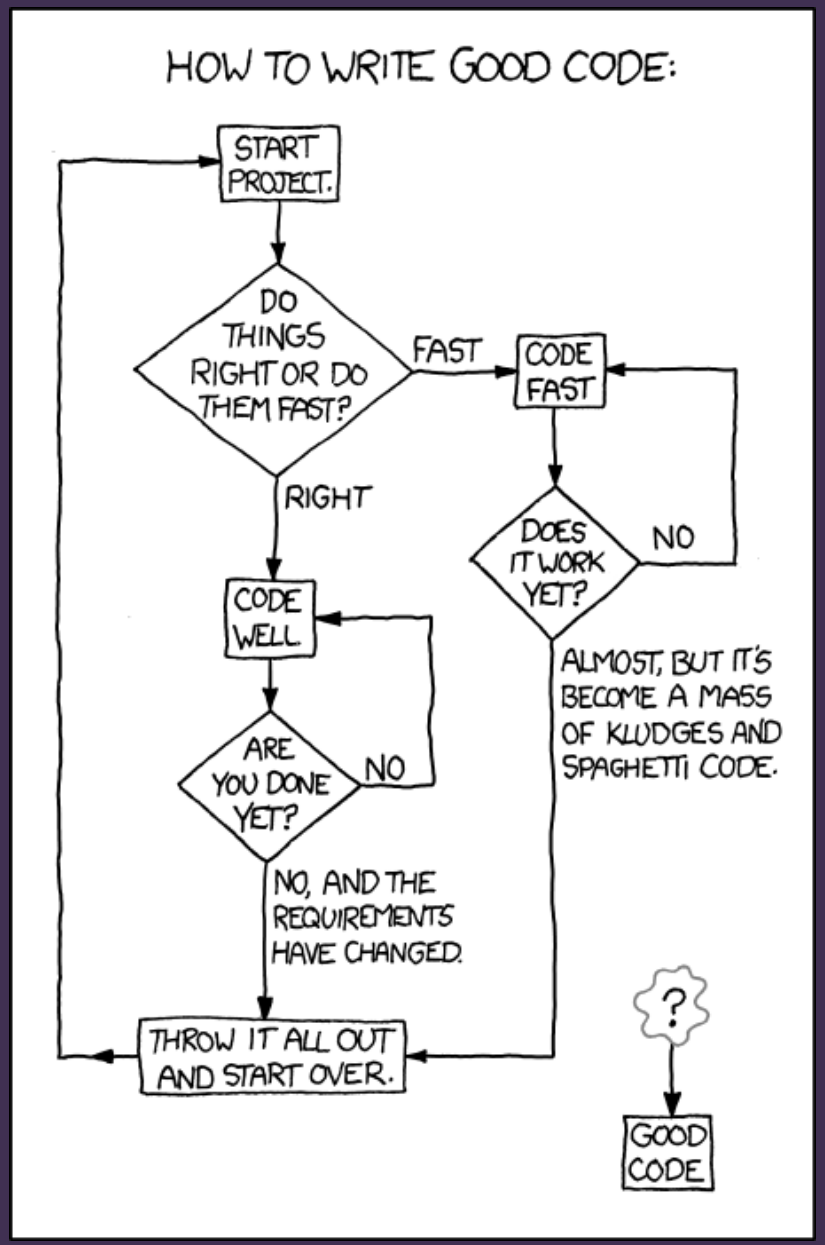
- Allow for common computations to have **common solutions**
- **Algorithm strategies** give widely applicable approaches for solving problems
- Can **prove** mathematically that an algorithm does what it is supposed to
- Can reason about the **complexity** (time, space etc) of an algorithm — and place **lower bounds** on the best possible algorithm

Why algorithms?

- Allow for common computations to have **common solutions**
- **Algorithm strategies** give widely applicable approaches for solving problems
- Can **prove** mathematically that an algorithm does what it is supposed to
- Can reason about the **complexity** (time, space etc) of an algorithm — and place **lower bounds** on the best possible algorithm
- **Computability** theory lets us reason about what computations are and are not possible

FALMOUTH
UNIVERSITY

Flowcharts



http://xkcd.com/844/

Flowcharts

Flowcharts

- Represent **control flow** in an algorithm or a computing system

Flowcharts

- Represent **control flow** in an algorithm or a computing system
- Start at the start, and follow the arrows!

Flowcharts

- Represent **control flow** in an algorithm or a computing system
- Start at the start, and follow the arrows!
- **Branches** (`if` statements) are represented as a choice of 2 or more arrows to go down

Flowcharts

- Represent **control flow** in an algorithm or a computing system
- Start at the start, and follow the arrows!
- **Branches** (`if` statements) are represented as a choice of 2 or more arrows to go down
- **Loops** are represented by the path looping back on itself

Flowcharts

- Represent **control flow** in an algorithm or a computing system
- Start at the start, and follow the arrows!
- **Branches** (`if` statements) are represented as a choice of 2 or more arrows to go down
- **Loops** are represented by the path looping back on itself
- Basic **concurrency** can be represented with "fork" and "join" points
  - Fork: go down multiple arrows simultaneously
  - Join: wait for multiple incoming arrows to arrive

UML

UML

- Unified Modeling Language

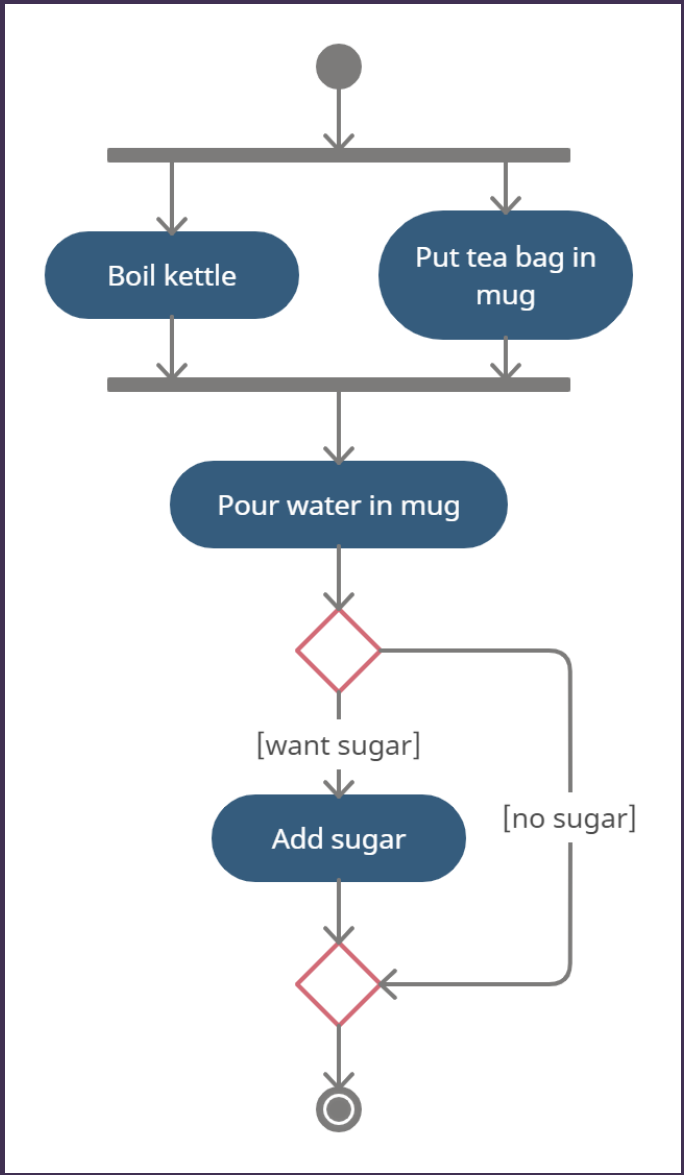
UML

- Unified Modeling Language
- Defines 14 types of diagram to represent various aspects of computing systems

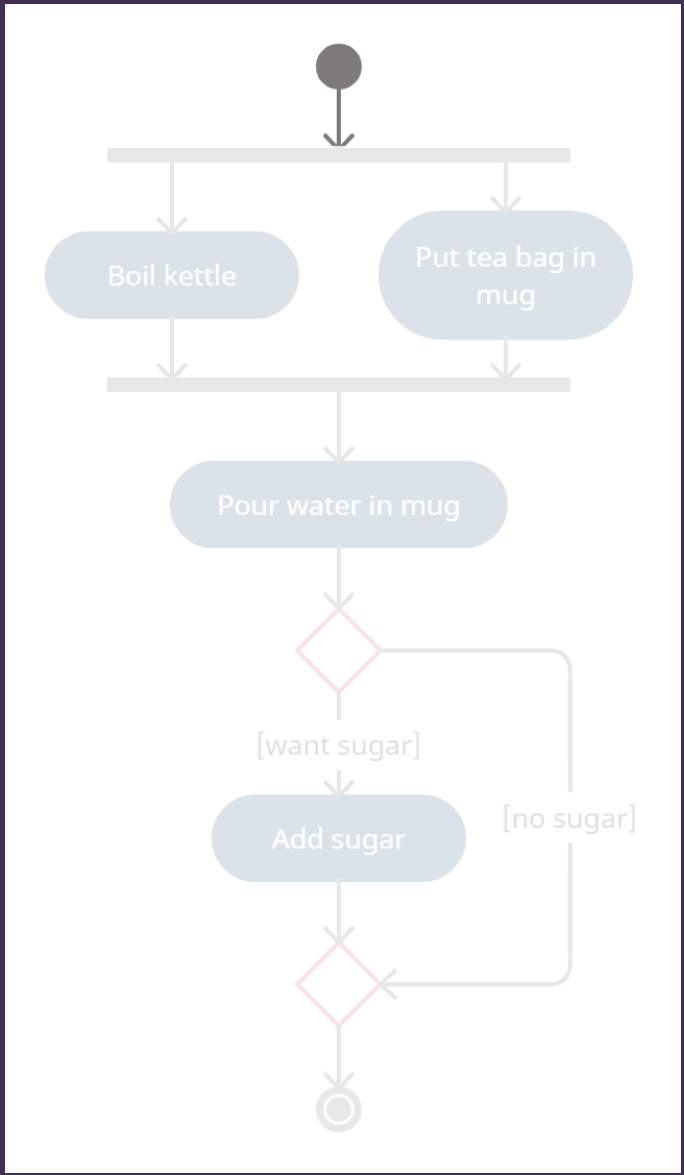
UML

- Unified Modeling Language
- Defines 14 types of diagram to represent various aspects of computing systems
- Activity diagrams: UML version of flowcharts

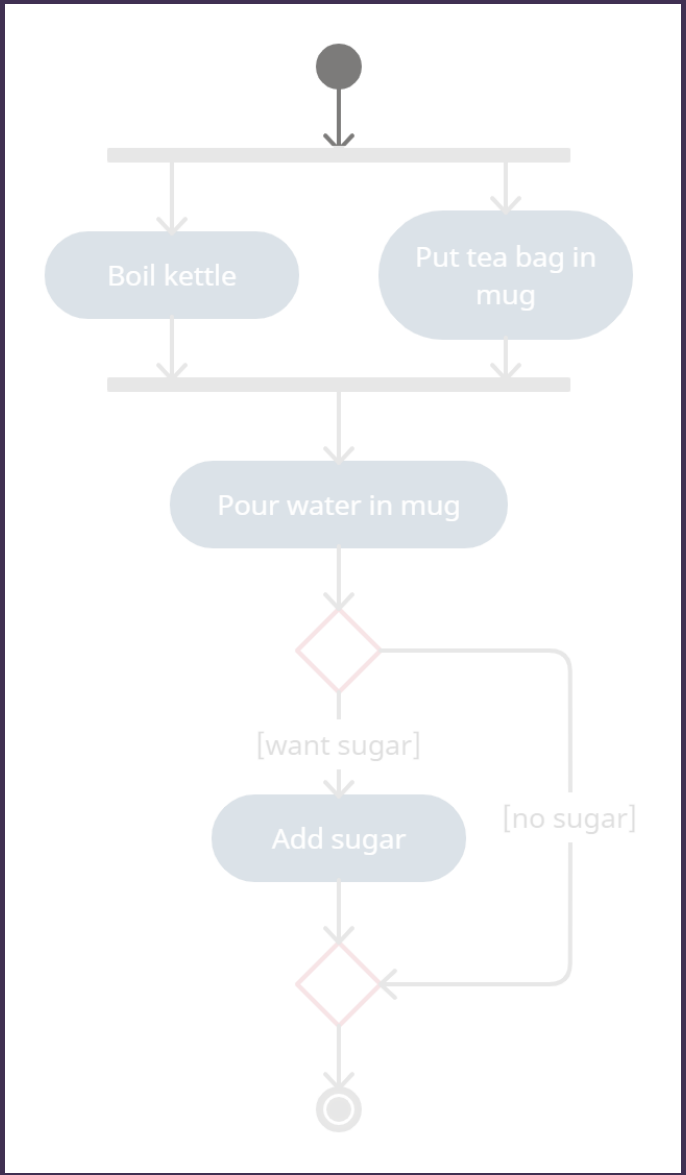
UML Activity Diagram



UML Activity Diagram Symbols

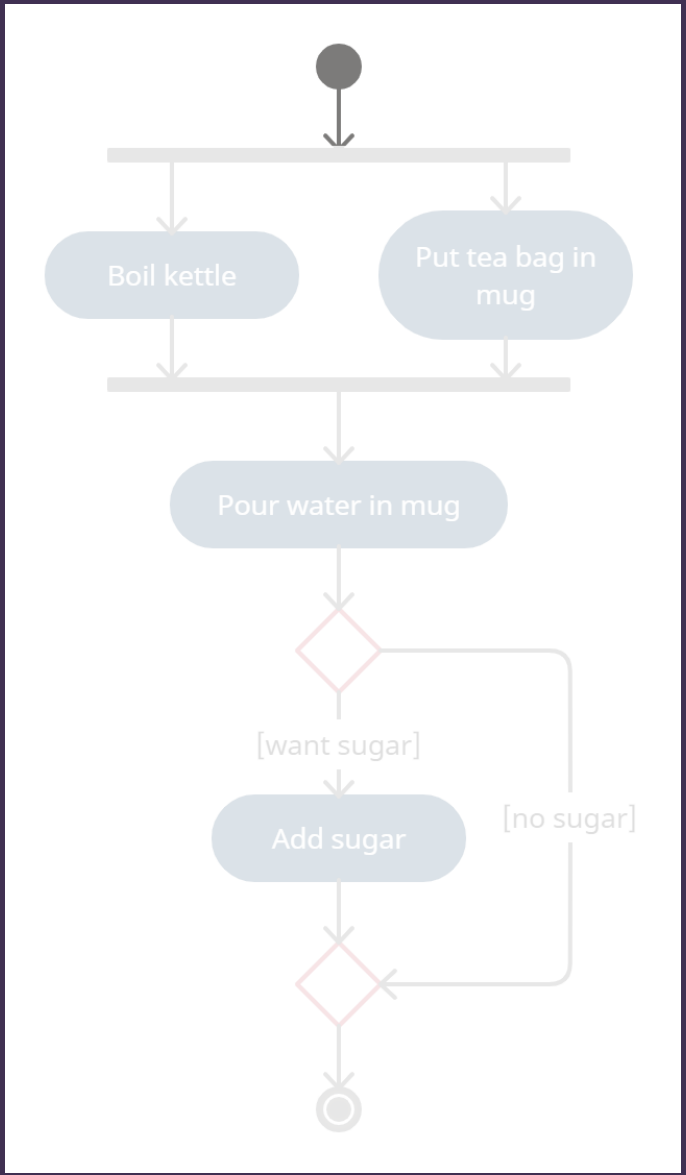


UML Activity Diagram Symbols



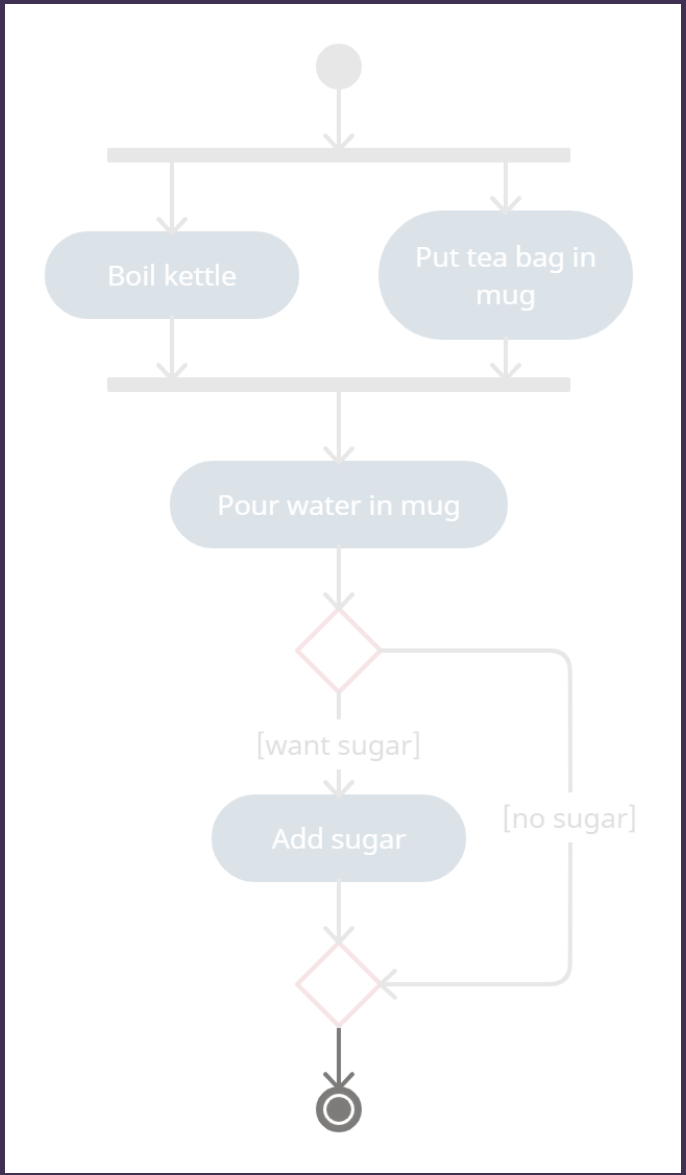
- **Start** node shows where control flow begins

UML Activity Diagram Symbols

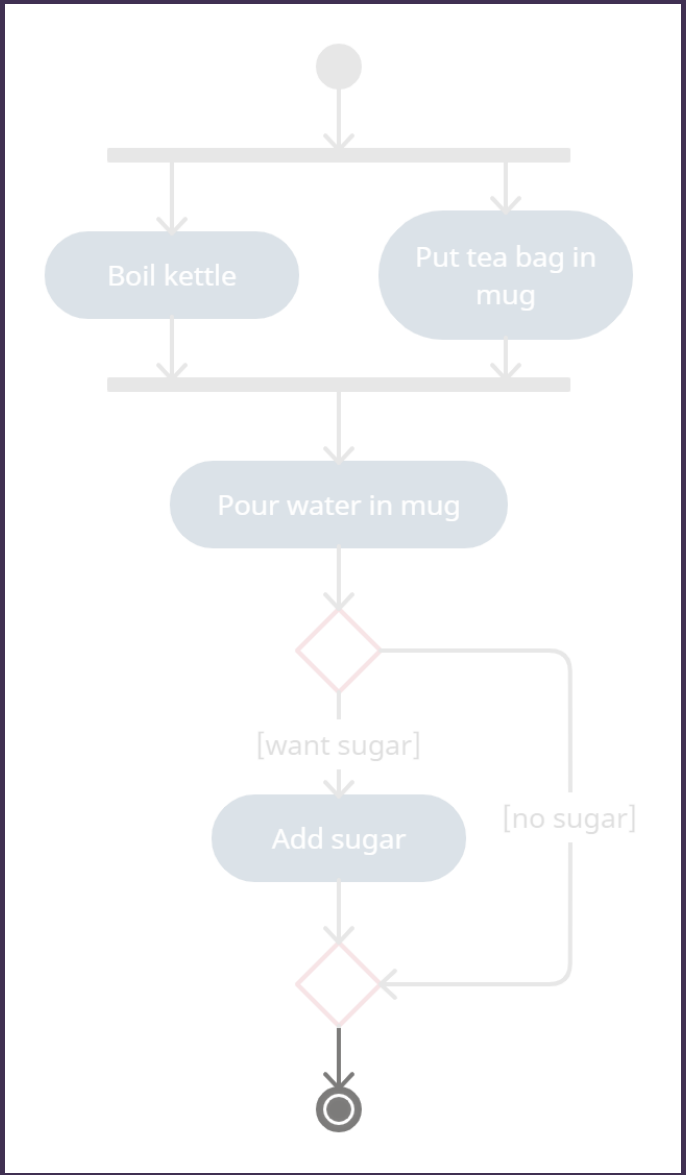


- **Start** node shows where control flow begins
- An activity diagram must have exactly one of these!

UML Activity Diagram Symbols

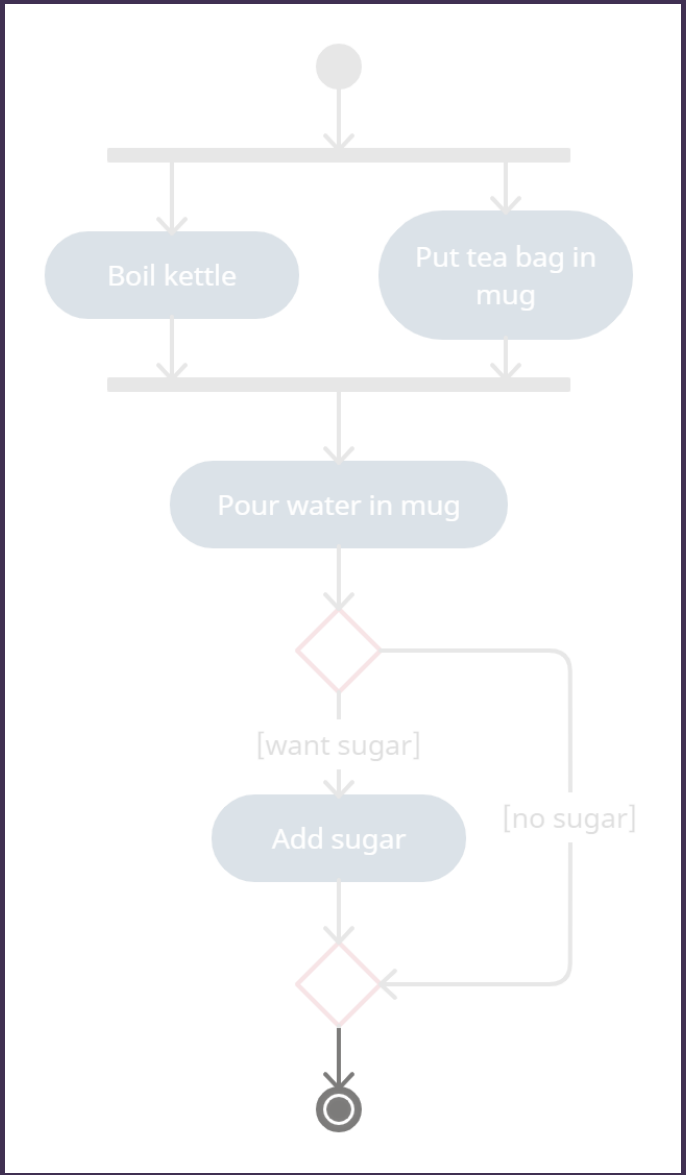


UML Activity Diagram Symbols



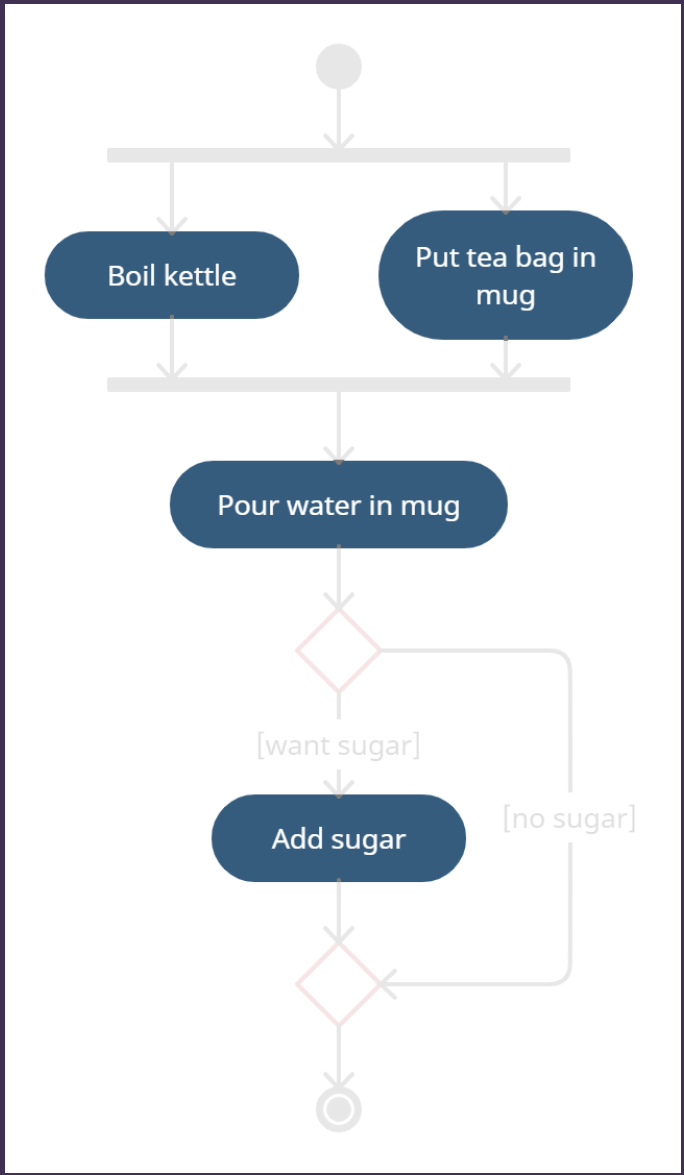
- **End** node shows where control flow terminates

UML Activity Diagram Symbols

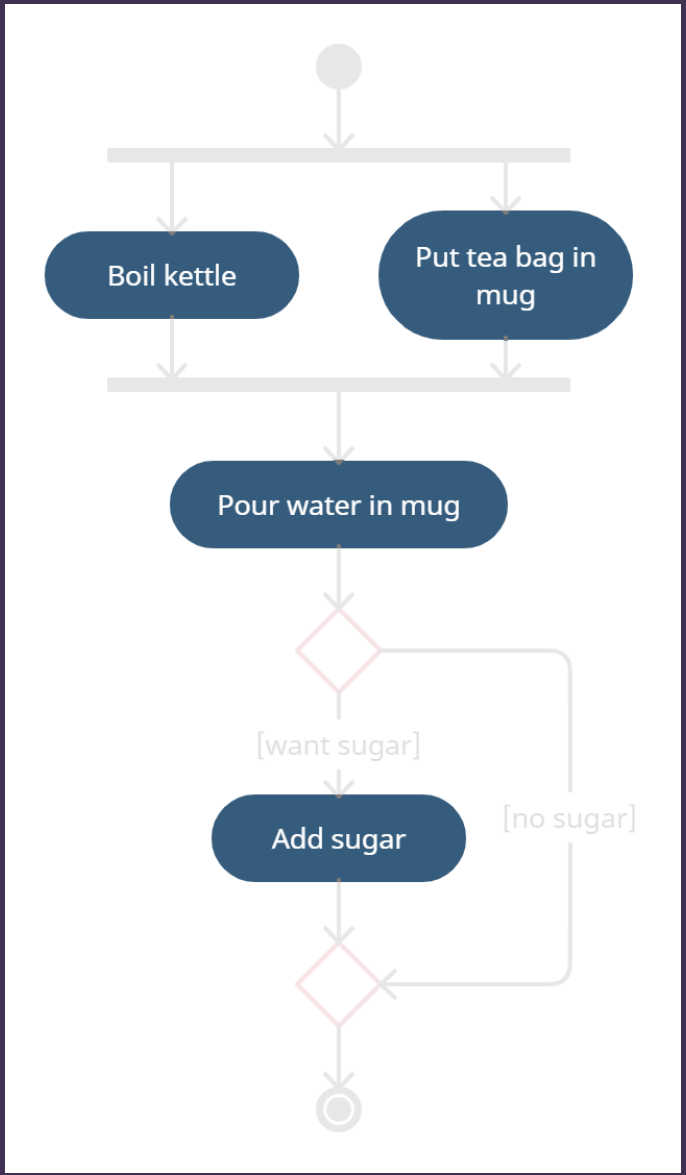


- **End** node shows where control flow terminates
- An activity diagram usually has one of these

UML Activity Diagram Symbols

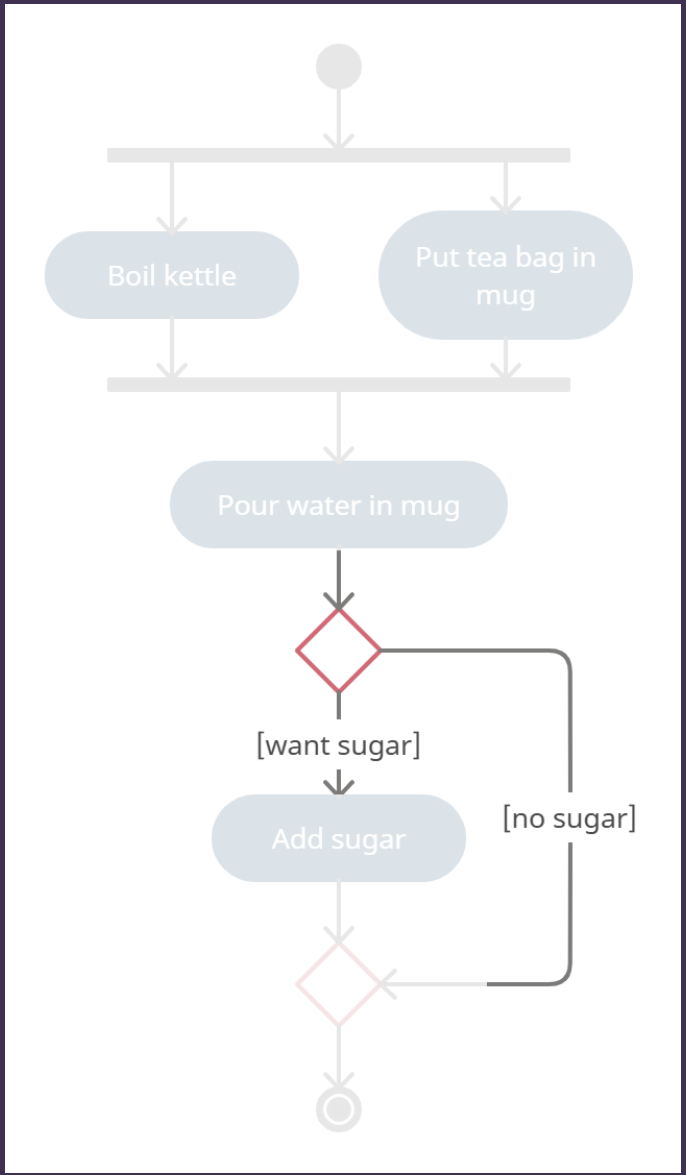


UML Activity Diagram Symbols

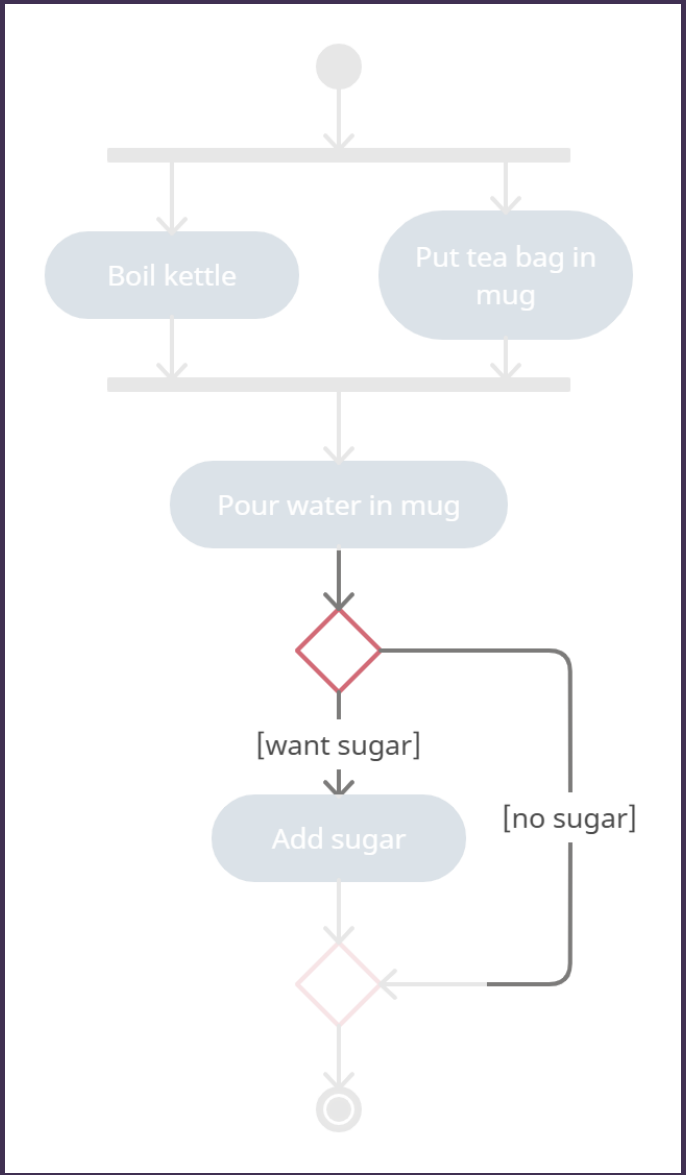


- **Action** or **activity** nodes describe the operations that are carried out

UML Activity Diagram Symbols

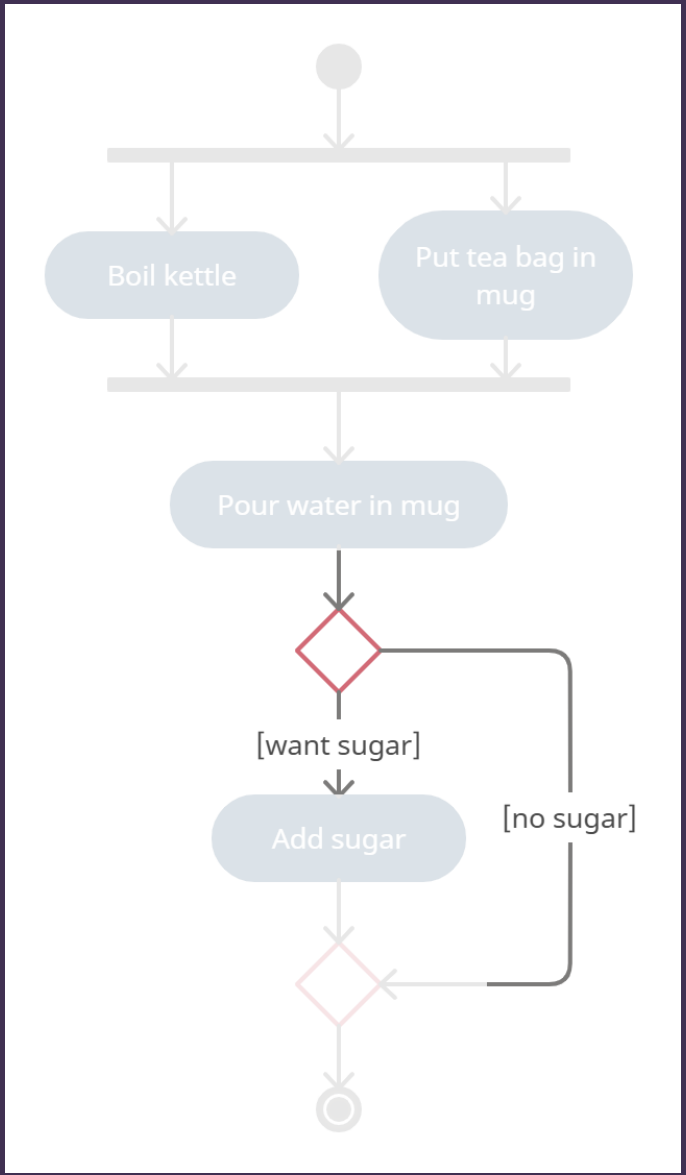


UML Activity Diagram Symbols



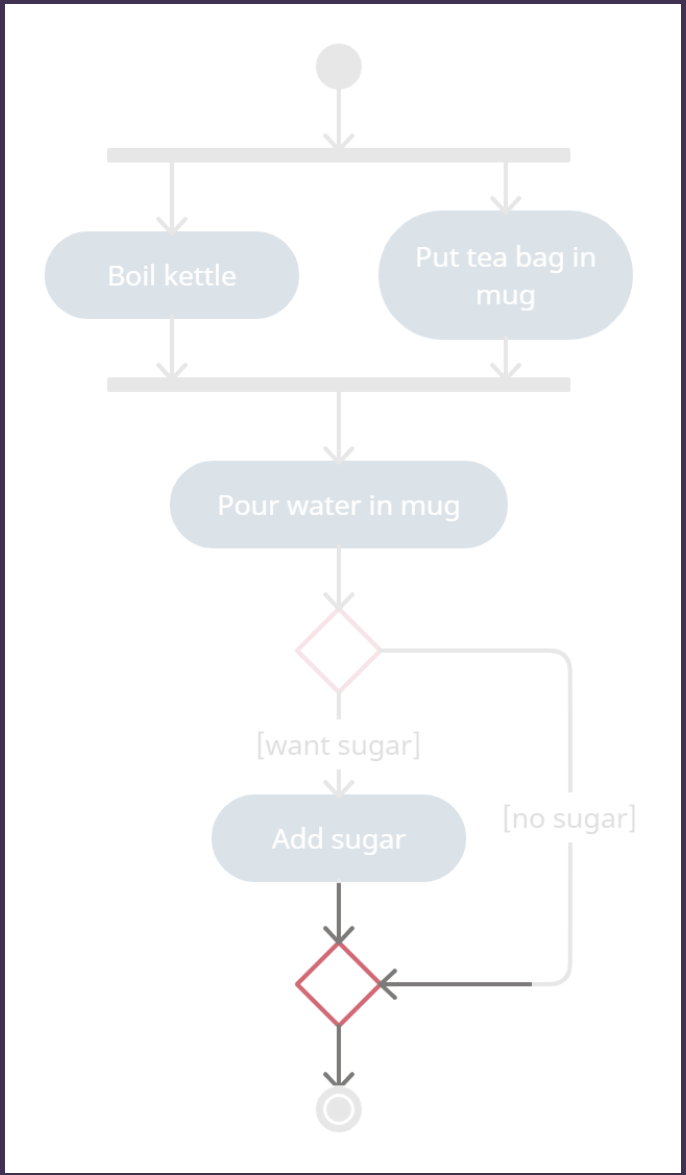
- **Decision** nodes represent a conditional branch (like an `if` statement)

UML Activity Diagram Symbols

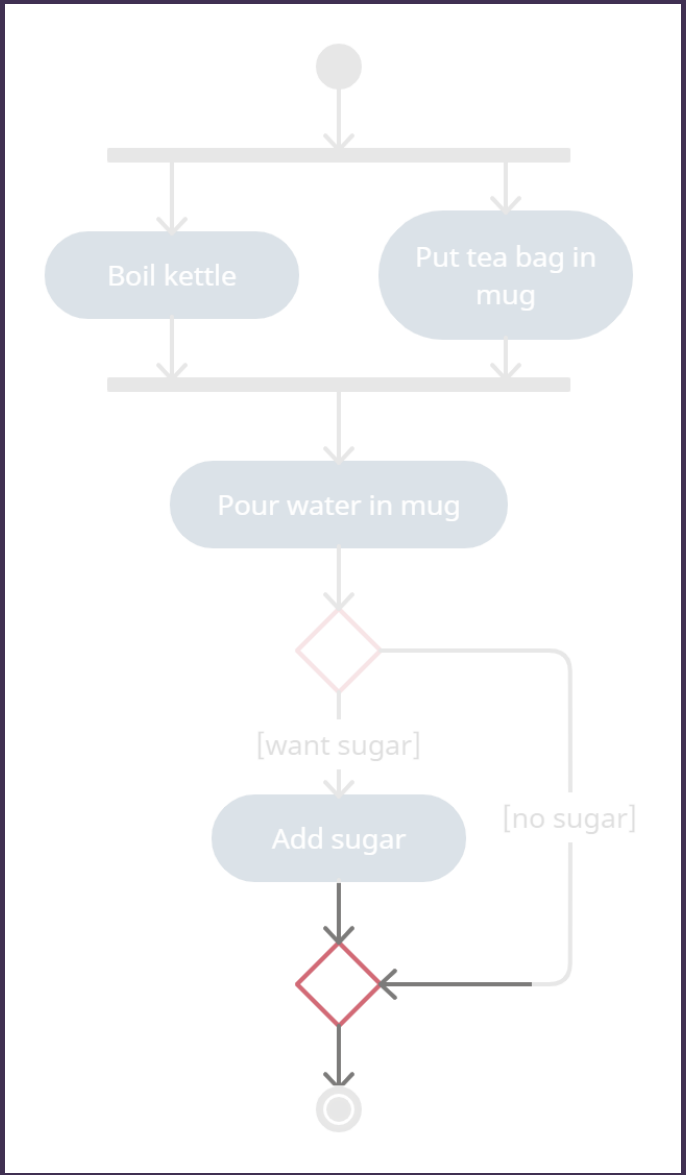


- **Decision** nodes represent a conditional branch (like an `if` statement)
- Outgoing arrows are labelled with descriptions of the conditions

UML Activity Diagram Symbols

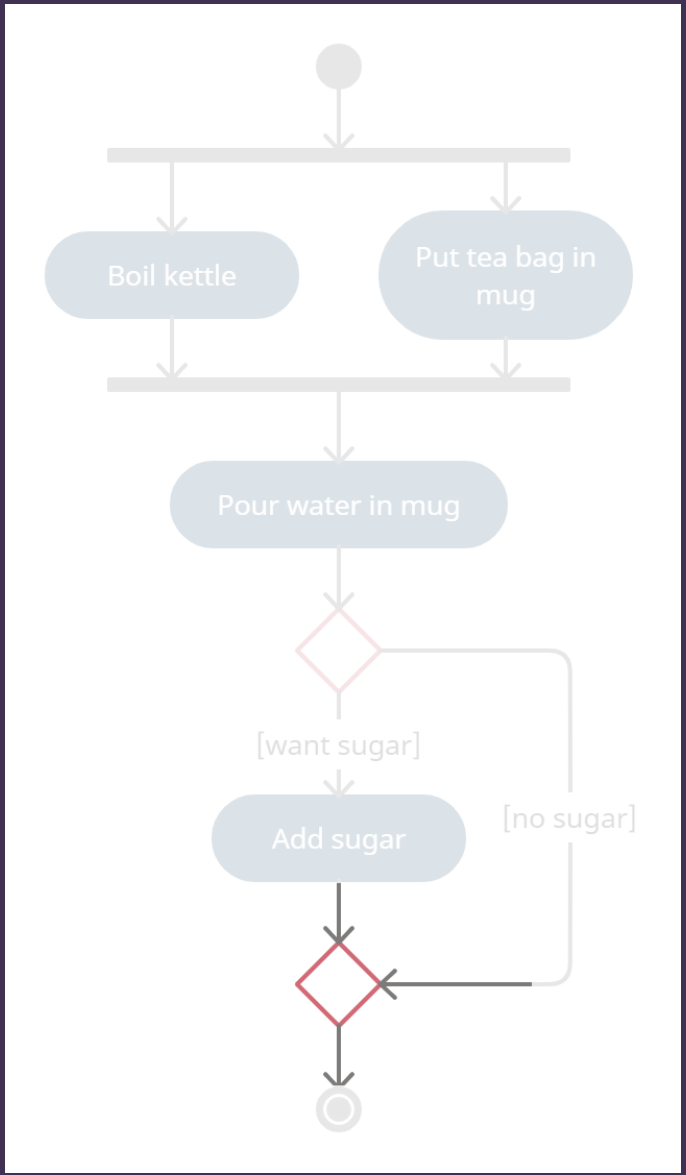


UML Activity Diagram Symbols



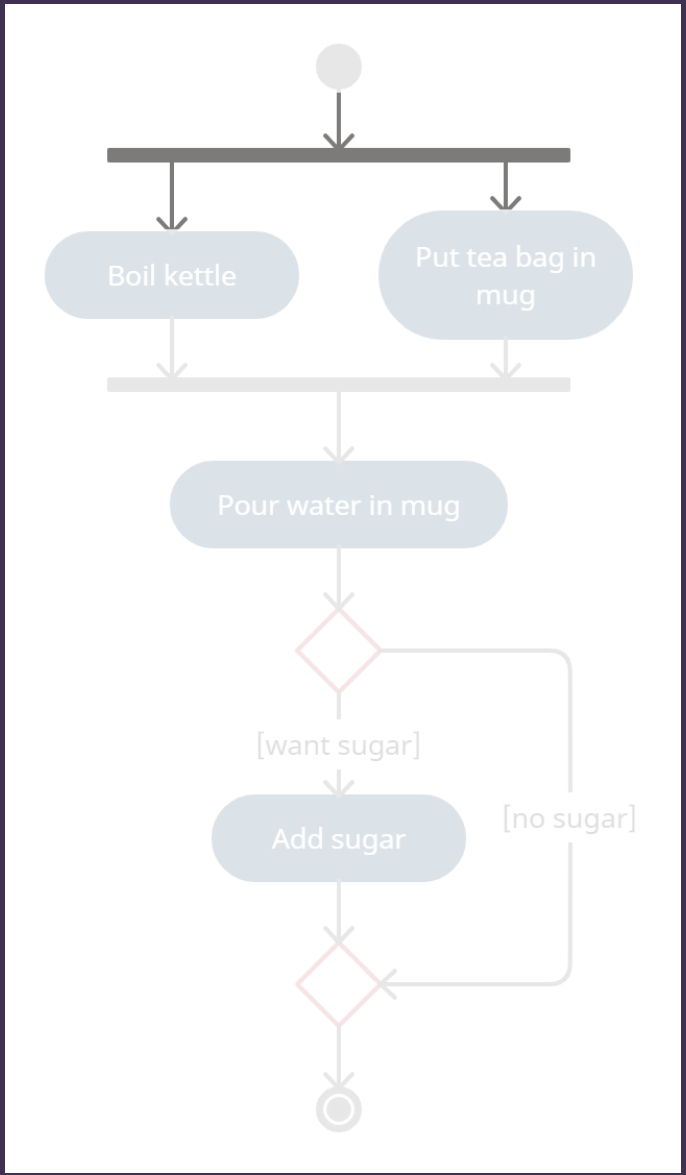
- **Merge** nodes allow alternative control paths to join together

UML Activity Diagram Symbols

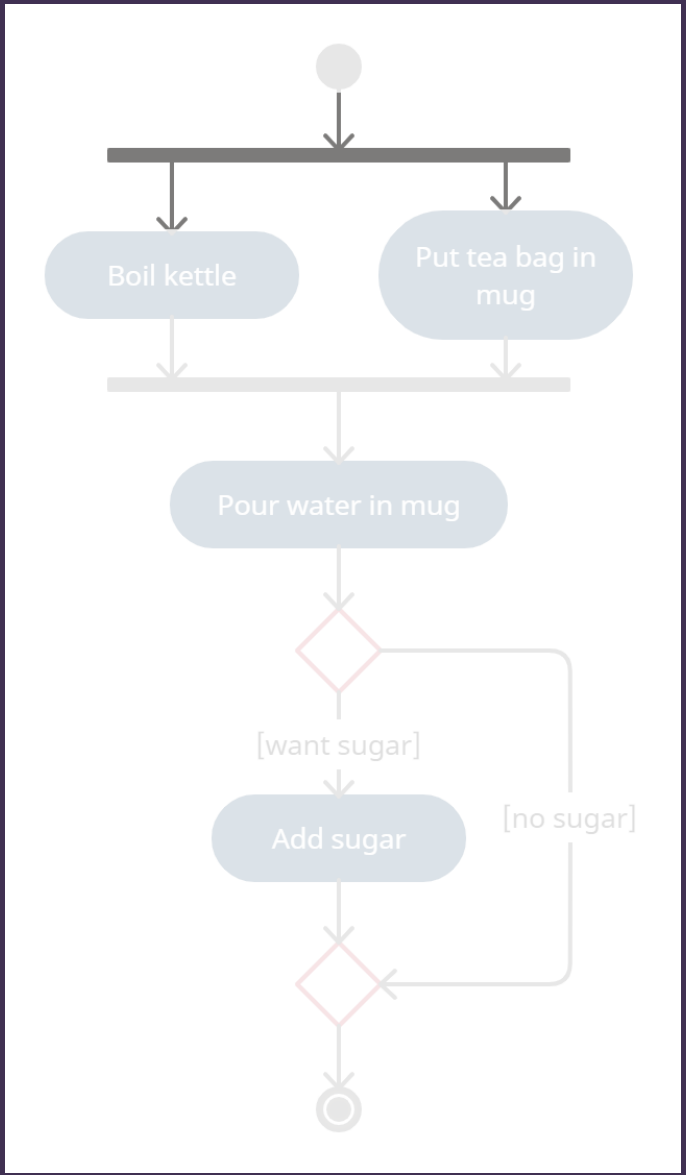


- **Merge** nodes allow alternative control paths to join together
- Commonly used after a decision node

UML Activity Diagram Symbols

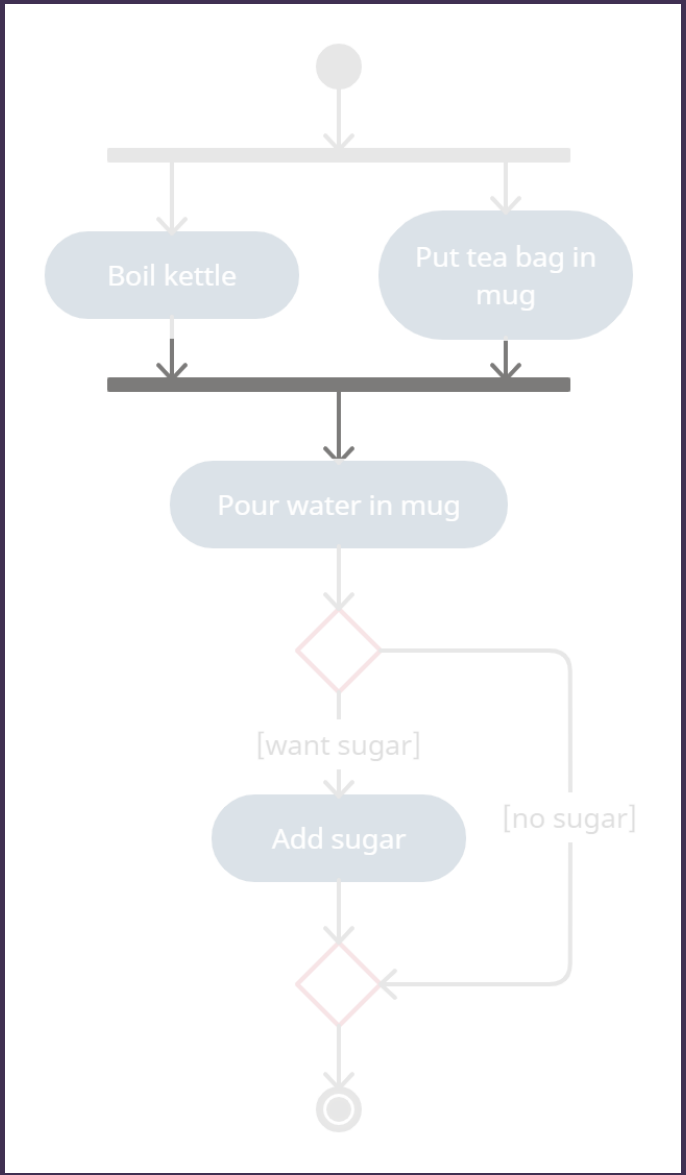


UML Activity Diagram Symbols

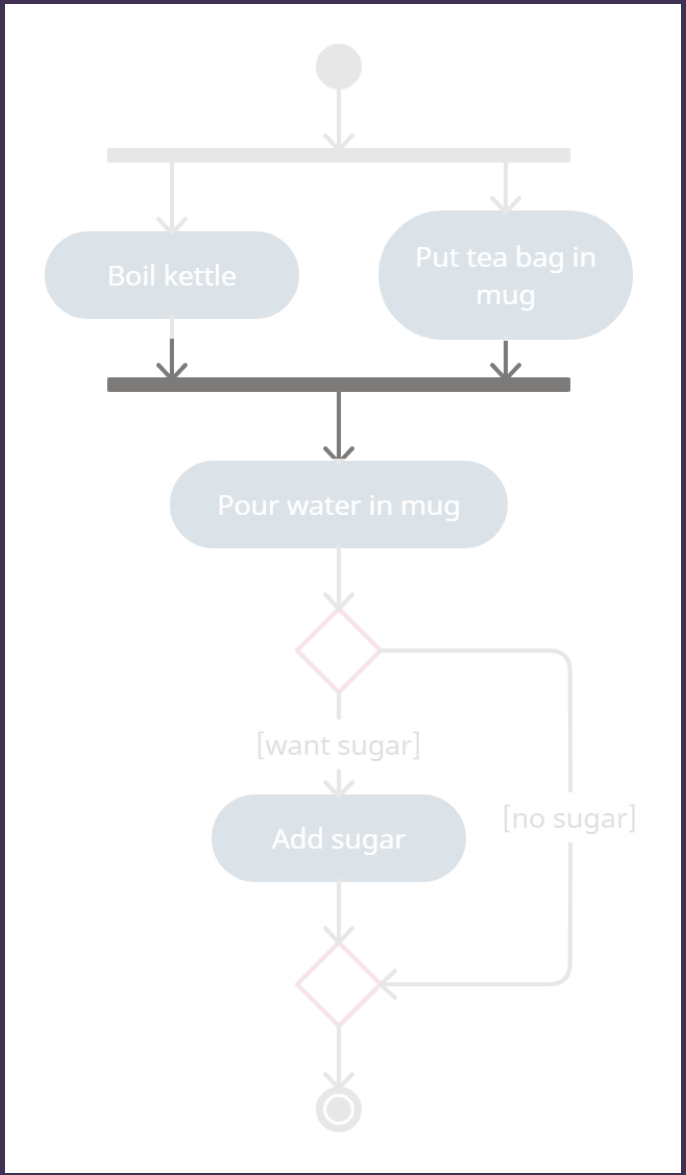


- **Fork** nodes represent concurrent (parallel) processes

UML Activity Diagram Symbols

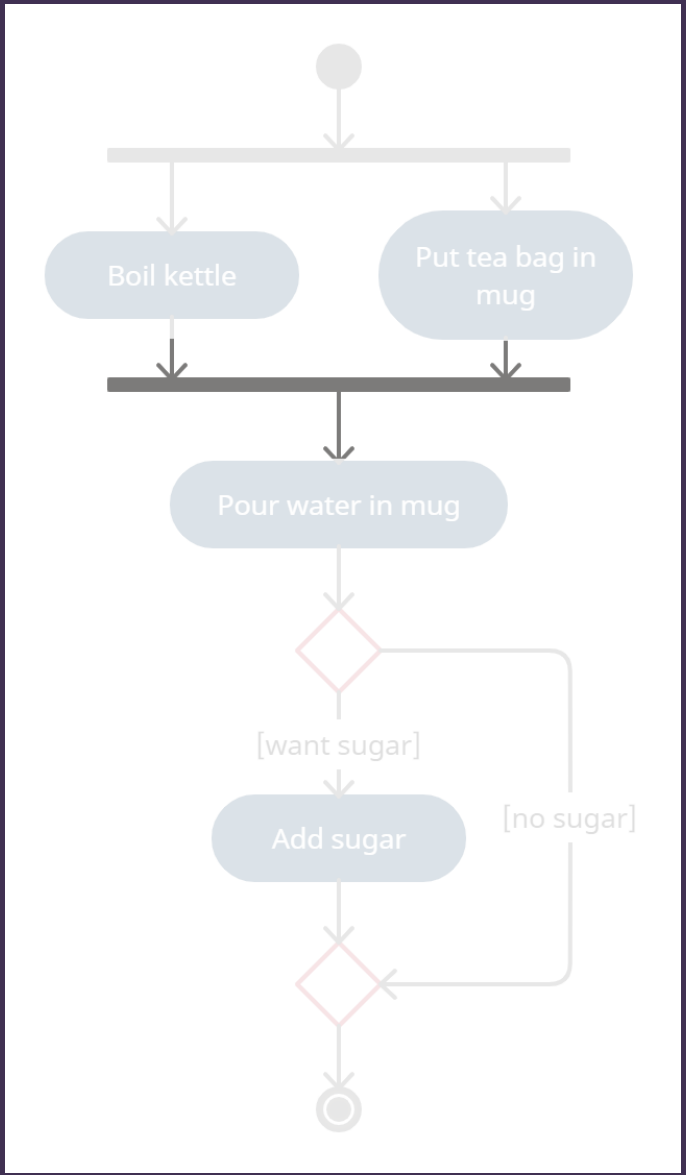


UML Activity Diagram Symbols



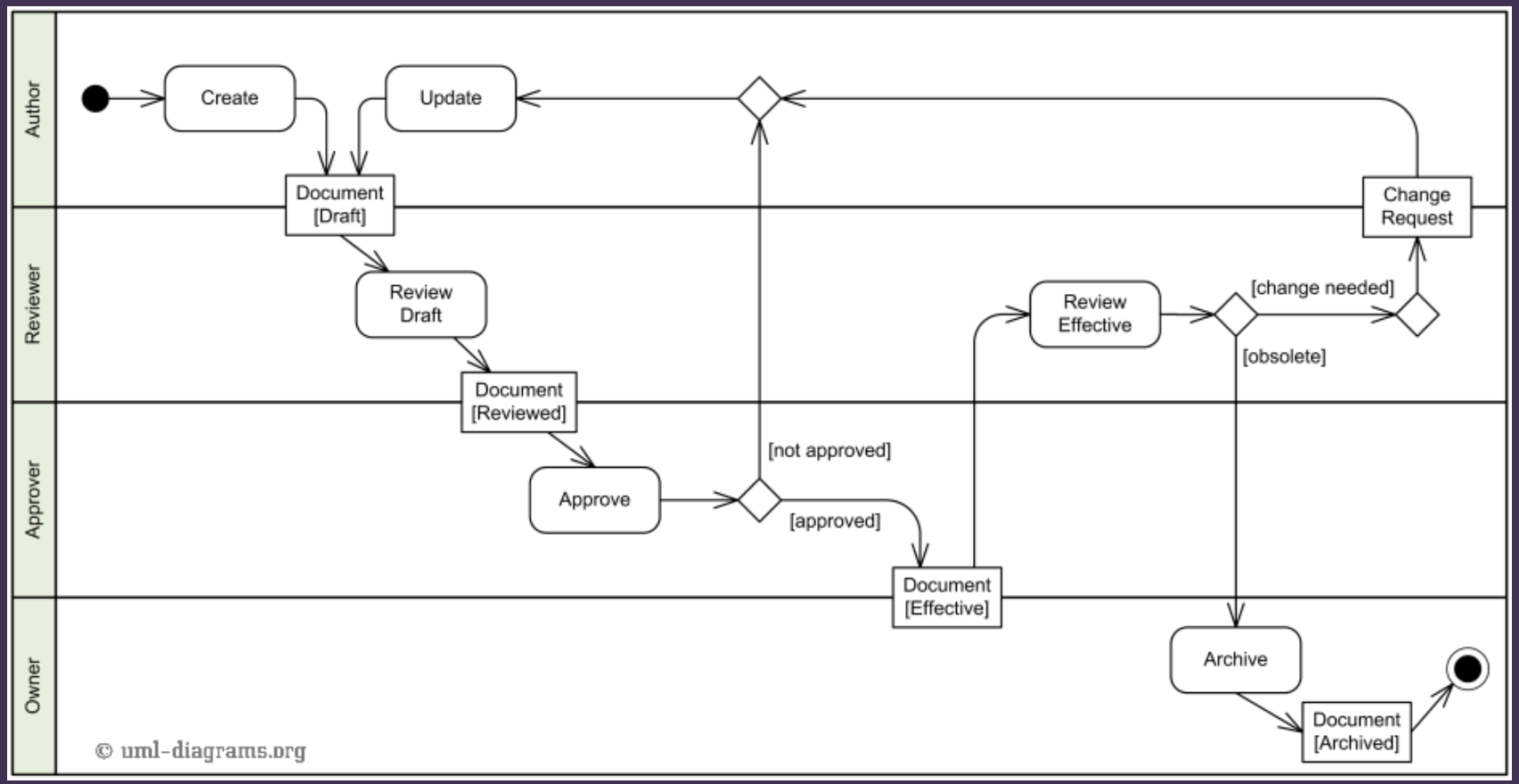
- **Join** nodes allow forked processes to join together again

UML Activity Diagram Symbols



- **Join** nodes allow forked processes to join together again
- Control waits until all processes are ready

Swimlanes



- Allow an activity diagram to represent interacting **subsystems**

Software for drawing UML diagrams

- Diagrams.net
- Creately
- Microsoft Visio
- (If you must) any other graphics software

Activity

- In your **breakout groups**
- **Draw** an activity diagram for **logging into Facebook**
- (Tip: either use the Teams whiteboard, or screen sharing with remote control, to collaborate on the same diagram)
- Include at least two swimlanes: **the user's browser/device** and **the Facebook server**

FALMOUTH
UNIVERSITY

Pseudocode

Pseudocode

Pseudocode

Flowcharts and activity diagrams are useful, but...

Pseudocode

Flowcharts and activity diagrams are useful, but...

- Can be time-consuming to draw

Pseudocode

Flowcharts and activity diagrams are useful, but...

- Can be time-consuming to draw
- Do not reflect structured programming concepts well

Pseudocode

Flowcharts and activity diagrams are useful, but...

- Can be time-consuming to draw
- Do not reflect structured programming concepts well

Pseudocode expresses an algorithm in a way that looks more like a structured program

Pseudocode example

```
print "How old are you?"
read age
if age < 13 then
    print "You are a child"
else if age < 18 then
    print "You are a teenager"
else
    print "You are an adult"
end if
```

Pseudocode example

$sum \leftarrow 0$ ▷ initialisation
for i in $1, \ldots, 9$ **do**
    $sum \leftarrow sum + i$
end for
print sum ▷ print the result

`https://socrative.com`, room code `FALCOMPED`:
what would this print?

Pseudocode example

$a \leftarrow 1$ ▷ initialisation
while $a < 100$ **do**
    $a \leftarrow a \times 2$
end while
print a ▷ print the result

`https://socrative.com`, room code `FALCOMPED`:
what would this print?

Formatting pseudocode

Formatting pseudocode

- Pseudocode is a **communication tool**, not a **programming language**

Formatting pseudocode

- Pseudocode is a **communication tool**, not a **programming language**
- Important: **clear**, **concise**, **unambiguous**, **consistent**

Formatting pseudocode

- Pseudocode is a **communication tool**, not a **programming language**
- Important: **clear**, **concise**, **unambiguous**, **consistent**
- **Not** important: adhering to a strict set of style guidelines, ensuring direct translatability to your chosen programming language

Level of abstraction

Level of abstraction

Whether working with flowcharts or pseudocode, choose your **level of abstraction** carefully

Level of abstraction: Good

Fill kettle
Turn kettle on
Put teabag in mug
if sugar wanted **then**
    Add sugar
end if
Wait for kettle to boil
if milk wanted **then**
    Pour water to $\frac{4}{5}$ full
    Add milk
else
    Fill mug with water
end if
Stir

Level of abstraction: Not so good

Position kettle beneath tap
Turn tap on
while water is below halfway point **do**
    Wait
end while
Turn tap off
Place kettle on base
Press power button
...

Level of abstraction: Silly

Place right palm on kettle handle
Bend fingers on right hand
Lift arm upwards
while tap spout is not directly above kettle **do**
    Move arm to the right
end while
Place left palm on tap handle
Bend fingers on left hand
Rotate left hand
...

Level of abstraction: also silly

Make a cup of tea

Activity

A number guessing game: The computer chooses a number between 1 and 20 at random. The player guesses a number. The computer says whether the guessed number is "too high", "too low" or "correct". The game ends when the correct number is guessed, or after 5 incorrect guesses.

- In your **breakout groups**
- **Write** pseudocode for the number guessing game