intro to software engineering

tech

PRELUDE
//
who the hell are you
what is this

t4tech

Twitter @ t4technyc

t4technyc@gmail.com

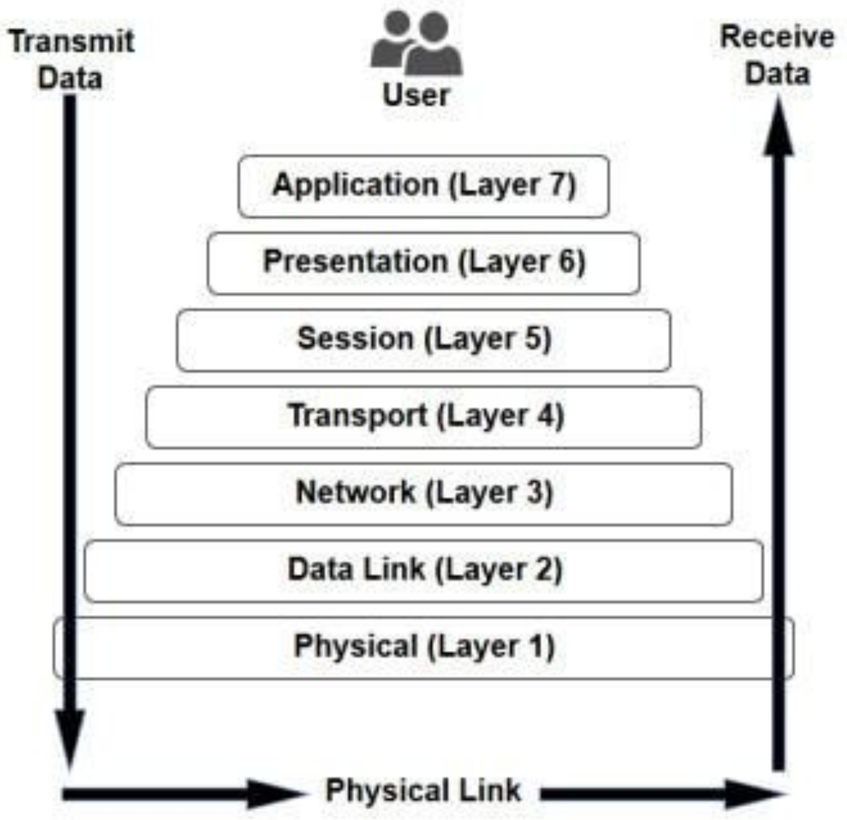
t4tech@protonmail.com

//

daly

ACT I
//
what the fuck is software

The 7 Layers of OSI

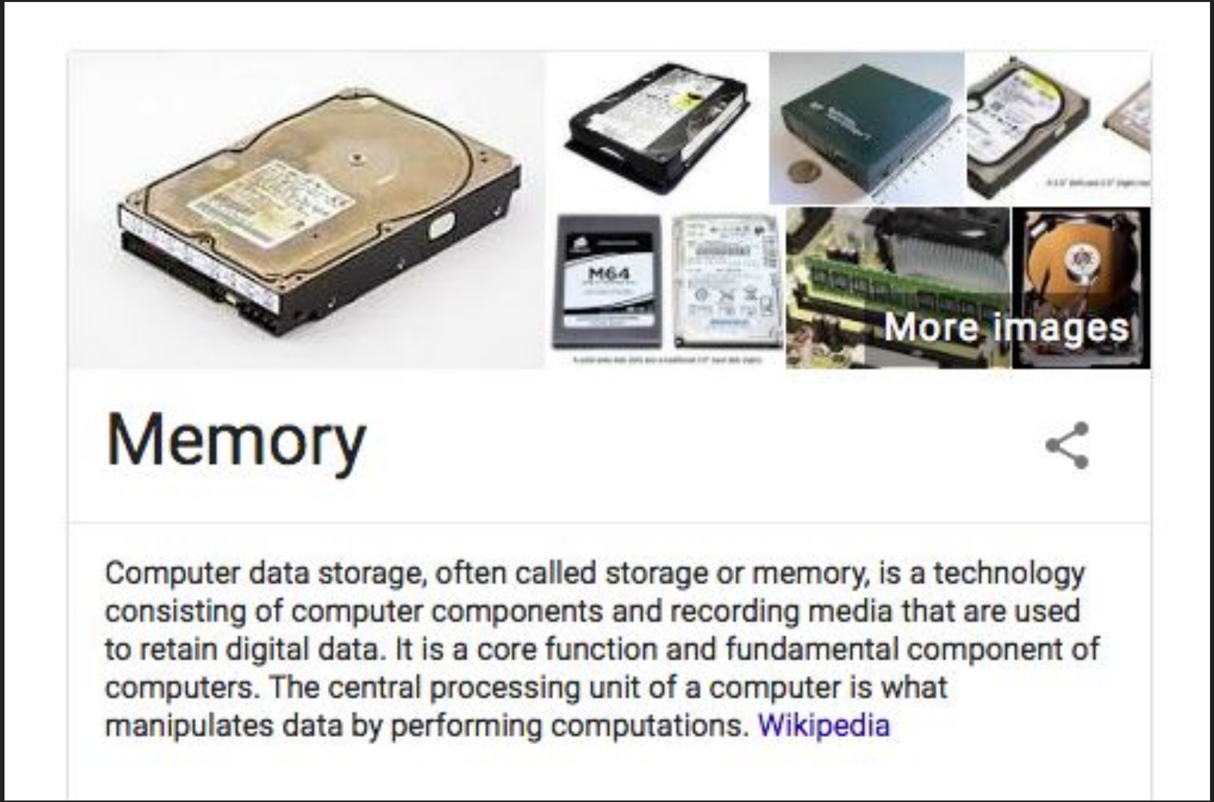


memory

data

algorithm

memory

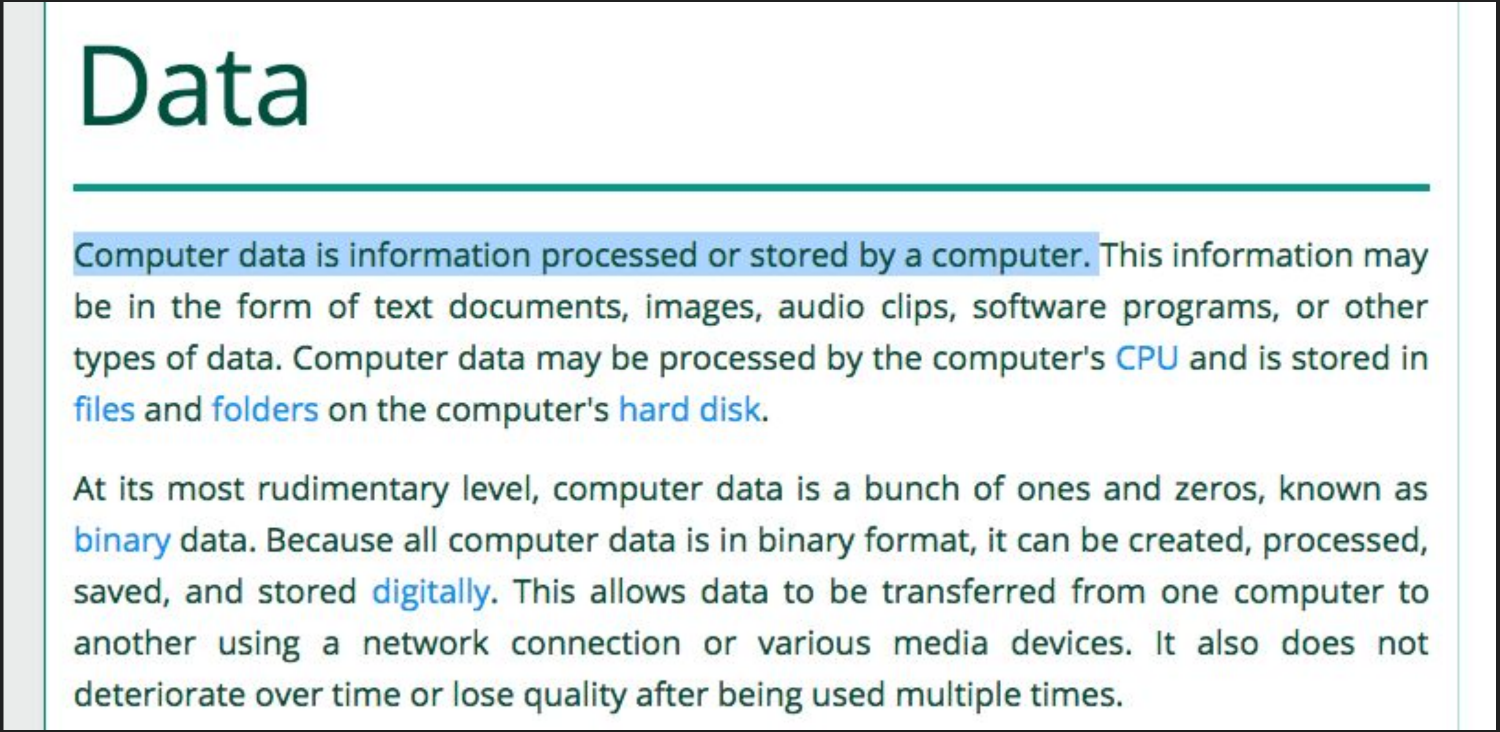


Memory

Computer data storage, often called storage or memory, is a technology consisting of computer components and recording media that are used to retain digital data. It is a core function and fundamental component of computers. The central processing unit of a computer is what manipulates data by performing computations. Wikipedia

thanks wikipedia

data

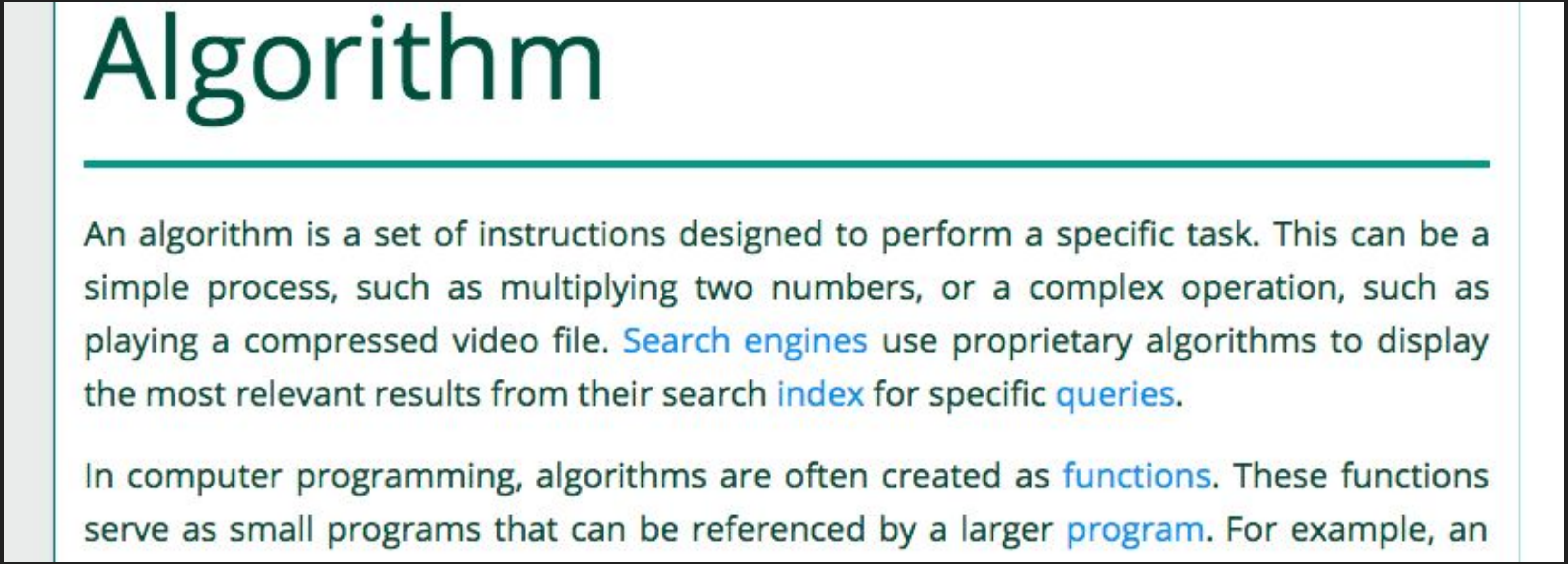


Data

Computer data is information processed or stored by a computer. This information may be in the form of text documents, images, audio clips, software programs, or other types of data. Computer data may be processed by the computer's CPU and is stored in files and folders on the computer's hard disk.

At its most rudimentary level, computer data is a bunch of ones and zeros, known as binary data. Because all computer data is in binary format, it can be created, processed, saved, and stored digitally. This allows data to be transferred from one computer to another using a network connection or various media devices. It also does not deteriorate over time or lose quality after being used multiple times.

thanks techterms.com



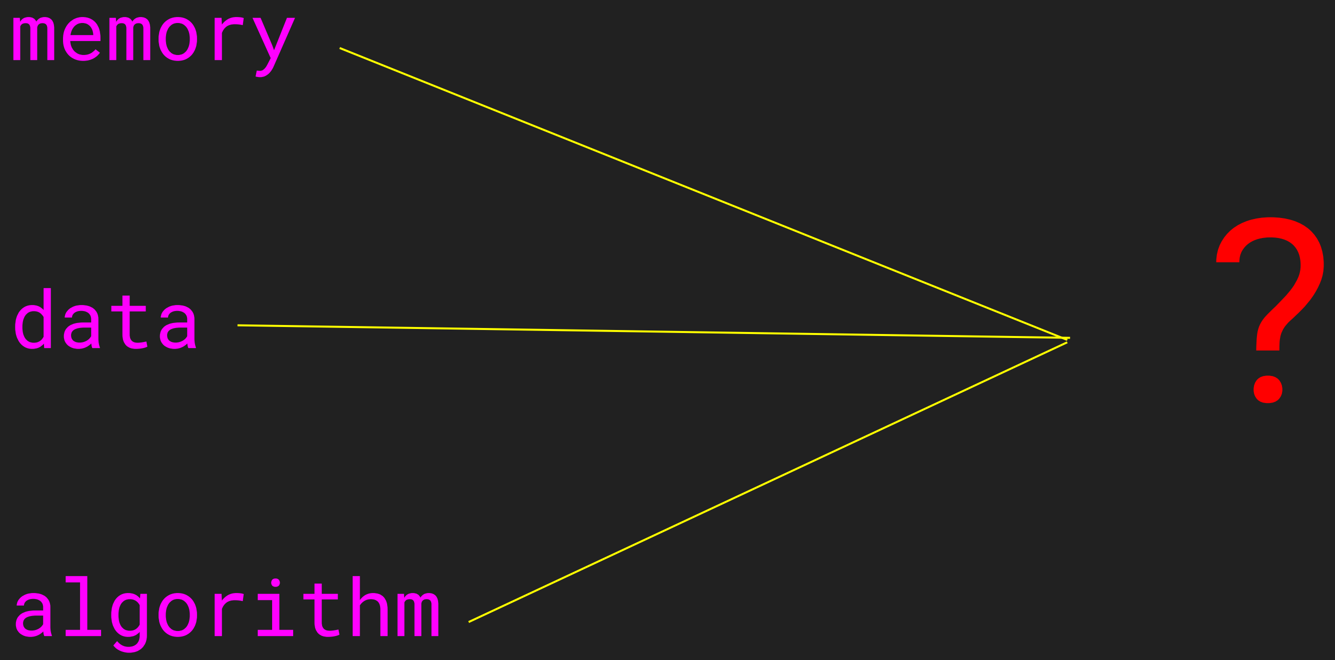
Algorithm

An algorithm is a set of instructions designed to perform a specific task. This can be a simple process, such as multiplying two numbers, or a complex operation, such as playing a compressed video file. Search engines use proprietary algorithms to display the most relevant results from their search index for specific queries.

In computer programming, algorithms are often created as functions. These functions serve as small programs that can be referenced by a larger program. For example, an

algorithm

thanks techterms.com


memory
data
algorithm
?

ACT II
//
the basics

pseudocode

Somewhere between your spoken language and your programming language of choice. Describes the steps that your program (or algorithm) will take. Often used as a tool for programmers to draft how they will write the actual code, check the flow of logic, and describe their thinking to other programmers.

evolution of pseudocode

Get flour, milk, eggs, sugar, and baking powder. Mix all the dry ingredients in a bowl, then add the wet ones. Whisk together until smooth. Heat up pan with butter. Pour desired amount of batter onto pan, cook until underside is golden brown. Flip and cook the other side until golden brown. Serve and enjoy.

evolution of pseudocode

- Get flour, eggs, baking soda, sugar, butter, milk
- Mix together dry ingredients first,
    - Then add wet ingredients
    - Whisk until smooth
- Heat up pan
    - Pour enough mixture to coat the pan
    - Cook until underside is golden brown
        - Flip and cook the other side
    - Repeat with rest of batter
- Serve with maple syrup and butter

evolution of pseudocode

```
let ingredients = {
    dry: [‘flour’, ‘baking soda’, ‘sugar’],
    Wet: [‘milk’, ‘eggs’, ‘butter’]
}
let batter = mixDryIngredients(ingredients.dry)
                .then((dryMix) => {
                    mixWithWet(dryMix, ingredients.wet)
                })
if batter {
    fryPancakes(batter).then((pancakes) => eat(pancakes)
} else {
    alert(‘Hey! Don’t forget to mix the batter!’)
}
```

variable declaration

INGREDIENTS

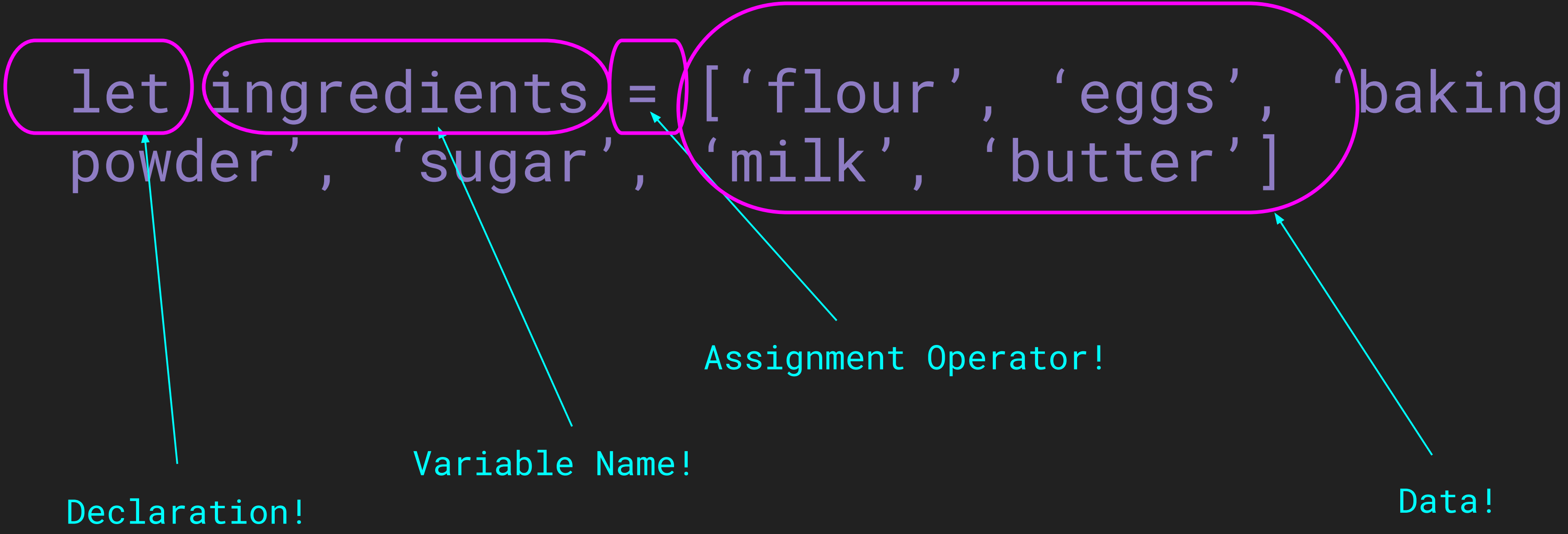
- Flour
- Eggs
- Baking Powder
- Sugar
- Milk
- Butter

```
let ingredients = [‘flour’,
‘eggs’, ‘baking powder’, ‘sugar’,
‘milk’, ‘butter’]
```

variable declaration

```
let ingredients = [‘flour’, ‘eggs’, ‘baking
powder’, ‘sugar’, ‘milk’, ‘butter’]
```

variable declaration



```
 let ingredients = [‘flour’, ‘eggs’, ‘baking
powder’, ‘sugar’, ‘milk’, ‘butter’]
```

data types

primitive

Character	String	Number	Boolean
‘a’	‘abcdefg’	123	true

data types

primitive

Character	String	Number	Boolean
‘a’	‘abcdefg’	123	true
‘2’	‘1’	1.4	false

data types

complex

Dictionary / Object

```
{
'abc': 1,
'123': 'abc',
'recipe': {...}
}
```

Array

```
[
'abc', 123, {}
]
```

Function

```
myFunction(num){
    let dub = num * 2
    return dub
}
```

data types

complex

Dictionary / Object

```
{
‘abc’: 1,
‘123’: ‘abc’,
‘recipe’: {...}
}
```

Array

```
[
‘abc’, 123, {}
]
```

Function

```
myFunction(num){
    let dub = num * 2
    return dub
}
```

Collections

Algorithm

ACT III
//
plinko & pinball
aka the hard (fun) stuff

control flow

conditionals & loops

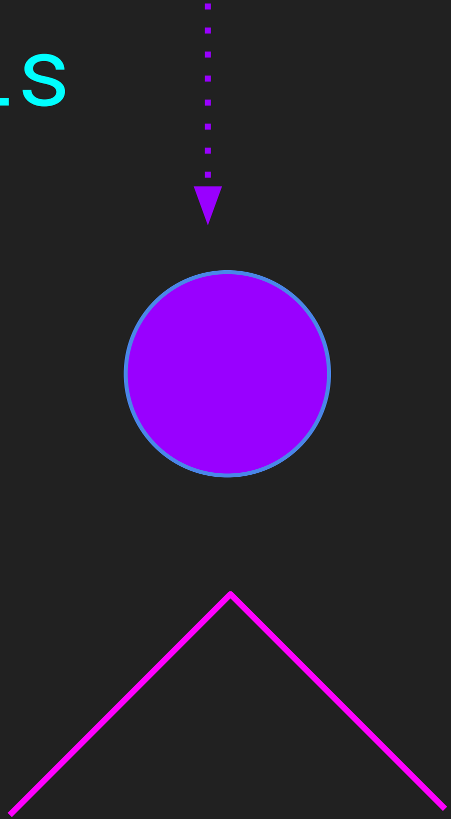
control flow

conditionals & loops

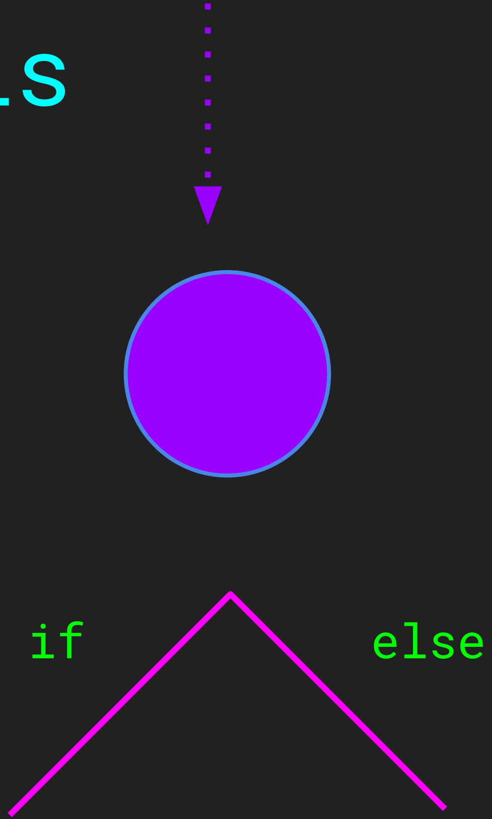
aka controlling the flow of logic in your code, enabling your software to make decisions on what to do next.

conditionals

conditionals



conditionals



conditionals

```
if ball is purple
    say ‘ball went right’
else
    say ‘ball went left’
```

conditionals

```
if ball == ‘purple’:
    print(‘ball went right’)
else:
    print(‘ball went left’)
```

conditionals

```
if ball == ‘purple’:
    print(‘ball went right’)
elif ball == ‘blue’:
    print(‘wrong ball color’)
else:
    print(‘ball went left’)
```

conditionals

```
if ball == None:
    ball = ‘red’
elif ball == ‘blue’:
    print(‘blue ball goes to the left’)
elif ball == ‘green’:
    print(‘green ball goes to the right’)
else:
    print(‘where\’d the ball go?’)
```

loops

leverage conditionals to repeat steps in your code until condition is met

loops

leverage conditionals to repeat steps in your code until condition is met

```
buy tub of horseradish hummus
dip chosen food vessel into hummus. Eat.
if tub is not empty
dip again
if tub is empty
weep
```

loops

leverage conditionals to repeat steps in your code until condition is met

```
let horseradishHummus = 10;
let eat = (food) => { food = food-1 }

while(horseradishHummus !== 0) {
    eat(horseradishHummus)
};

cry();
```

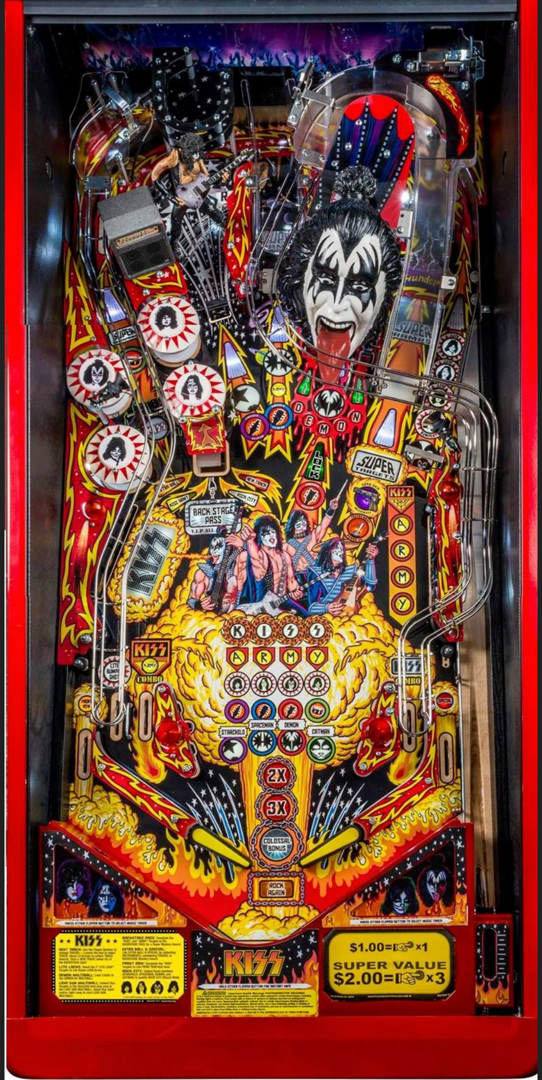
loops

leverage conditionals to repeat steps in your code until condition is met

```
for(let i in myCollection) {...}
while(i < myCollection.length) {...}
myCollection.forEach((thing) => {...})

for x in myCollection:
for x in range(len(myCollection)):

for (a = 0; a < myCollection; a++) {...}
```


KISS
BACK STAGE PASS
SUPER TARGETS
DEMON
LOCK
ARMY
COMBO
STARCHILD
SPACEMAN
DEMON
CATMAN
2X
3X
COLOSSAL BONUS
ROCK AGAIN
$1.00= x1
SUPER VALUE
$2.00= x3

BACK STAGE PASS
KISS
ARMY
DEMON
2X
3X
COLOSSAL BONUS
ROCK AGAIN
$1.00= x1
SUPER VALUE
$2.00= x3

EPILOGUE
//
the jargon

paradigms

Declarative: says what the program does (what)
Imperative: says how the program behaves (how)

Object Oriented: a bunch of things that talk to each other
Functional: a bunch of algorithms run in order

Event Driven: designed to listen for inputs and react with given outputs

concepts

Synchronous: steps of the code are run in order, one after the other.

- AKA “Blocking”

Asynchronous: steps of code are executed only when triggered, regardless of whether or not the previous step has finished.

- AKA “Non-Blocking”

Prototypal: An inheritance pattern where an instance of an object is a grandchild of an ancestor type, inheriting behaviors and traits from the top down

Classical Inheritance: An inheritance pattern where a blueprint is created for a type of object, wherein all instances of that object will have the same beginning traits and behaviors.

NEVER BE AFRAID
TO ASK FOR
DEFINITION OR
CLARIFICATION