IMMUTABILITY

Beau Harrison

WIKIPEDIA

In object-oriented and functional programming, an immutable object (unchangeable object) is an object whose state cannot be modified after it is created. This is in contrast to a mutable object (changeable object), which can be modified after it is created. In some cases, an object is considered immutable even if some internally used attributes change, but the object's state appears unchanging from an external point of view.

Strings and other concrete objects are typically expressed as immutable objects to improve readability and runtime efficiency in object-oriented programming. Immutable objects are also useful because they are inherently thread-safe. Other benefits are that they are simpler to understand and reason about and offer higher security than mutable objects.

EXAMPLES

RUST

memory safety and concurrency
no null pointers or shared mutable state

RUST

```
let x = 5; // immutable by default
x = 10; // compile time error
let mut y = 6; // we can now change y
y = 2;

let y: i32 = 16;
let y = y + 1; // shadowing, transforms y with immutability

let mut scores = HashMap::new();
scores.insert(String::from("Blue"), 10); // adding items
```

RUST

BORROWING

```
let v = vec![1,2,3];

fn foo(v: &Vec) {
  v.push(4); // compile time error
}

foo(&v); // the ref is immutable by default
```

RUST

BORROWING

```
let mut v = vec![1,2,3]; // the vector should be mutable

fn foo(v: &mut Vec) {
  v.push(4);
}

foo(&mut v); // as well as the ref
```

The Rust compiler only allows one borrower at a time, it can only be changed in one place at a time. Immutability by default makes this possible.

RUST

CONCURRENCY

Once again, immutability by default gives peace of mind that when multiple processes are using the same variable, it will not change.

PYTHON

Everything in Python is an object. An object's mutability is determined by its type. Some objects like lists and dictionaries are mutable, meaning you can change their content without changing their identity. Other objects like integers, floats, strings and tuples are objects that can not be changed.

PYTHON

```
>>> temp = "hello world"
>>> temp[0] = "b"
Traceback (most recent call last):
  File "", line 1, in
TypeError: 'str' object does not support item assignment
```

PYTHON

```
>>> temp = (10, 20, 30)
>>> temp[0] = 40
Traceback (most recent call last):
  File "", line 1, in
TypeError: 'tuple' object does not support item assignment
```

WHY DOES IT MATTER?

- Memory usage
- Performance
- Peace of mind
- Readability