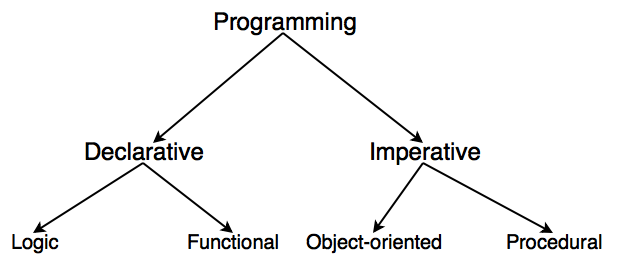
**1. 함수형 프로그래밍, 절차지향 프로그래밍, 객체지향프로그래밍이란 무엇인가**

**2. 함수형 프로그래밍이 갖는 장점과 단점은 무엇인가.**

**3. 순수함수란 무엇인가.**

**Programming**은 크게 다음과 같이 분류된다.

출처:

https://subscription.packtpub.com/book/application\_development/9781788996648/4/ch04lvl1sec20/functional-programming

**Declarative**(선언형)과 **Imperative**(명령형)을 간단하게 비교하면 이러하다.

* 선언형: 무엇인가를 작업하기 위해 어떻게 진행할 것인지를 나열한다. (컵라면을 끓인다.)
* 명령형: 무엇인가를 작업하기 위한 방법을 정의한다. (뚜껑을 연다 -> 물을 붓는다 -> 면이 익을 때까지 기다린다.)

선언형은 다시 **Procedural**(절차지향)과 **Object Oriented**(객체지향)으로 나뉘는데,

|  |  |
| --- | --- |
| https://k.kakaocdn.net/dn/bP0W42/btqFnO1Ueoa/zih4vrCYzYy5VJclKJ1kQ0/img.jpg | https://k.kakaocdn.net/dn/kyhoj/btqFp51IKf9/5LryieovRu66M9kNOKpz51/img.jpg |
| [출처: https://m.blog.naver.com/PostView.nhn?blogId=atalanta16&logNo=220249264429&proxyReferer=https:%2F%2Fwww.google.com%2F](https://m.blog.naver.com/PostView.nhn?blogId=atalanta16&logNo=220249264429&proxyReferer=https:%2F%2Fwww.google.com%2F) | |

### Procedural Programming(절차지향 프로그래밍)

Procedural Programming(절차지향 프로그래밍)은,

▶ 물이 위에서 아래로 흐르듯이 순차적으로 처리가 이루어지며, 프로그램 전체가 유기적으로 연결된다.

▶ 데이터를 중심으로 프로그램 구현

**▶**장점: 컴퓨터의 처리구조와 유사해 실행속도가 빠르다.

▶ 단점: - 실행순서가 정해져 있어, 코드의 순서가 바뀌면 동일한 결과를 보장하기 어렵다.

           - 유지보수가 어렵다.

           - 디버깅이 어렵다.

▶ 대표적 언어: C언어

### Object Oriented Programming(객체지향 프로그래밍)

Object Oriented Programming(객체지향 프로그래밍)은,

▶ 실제 세계를 추상화하여 객체 공통의 Property와 method를 추출한다.

ex) 휴대전화의 공통부분 : 통화한다(0), 사진을 찍는다(0) / 삼성에서 만들었다(x) 폴더블이다(x)

▶ 데이터와 절차를 하나의 덩어리로 묶어서 생각한다. (위의 그림 참조)

▶ 기능을 중심으로 프로그램 구현

▶ 특징:

1. **Encapsulation** (캡슐화)

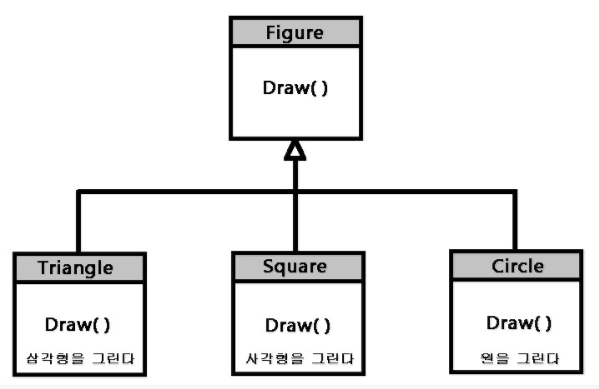
- 은닉성: 묶은 내용 중, 외부에서 접근하지 못하는 Property와 method를 만들 수 있다. 외부와의 상호작용은 method를 통해 이루어진다.

- 데이터 알고리즘(데이터를 다루는 방식)을 하나의 묶음으로 정리한다. (변수와 함수를 Class로 묶는다.)

2. **Inheritance** (상속)

- 부모 class의 Property와 method를 자식 class에서 사용할 수 있어 코드 재사용이 가능하고, 중복코드를 제거할 수 있으며, 물려 받은 Property와 method와 더불어 필요한 기능을 추가해 확장할 수도 있다.

3. **Polymorphism** (다형성)



출처: https://debugdaldal.tistory.com/152

- 하나의 class나 method가 다양한 방식으로 동작이 가능 한 것을 의미한다.

- Overriding (오버라이딩)

필요에 따라서, 하위 class에서 상속받은 상위 class의 method를 그대로 사용하지 않고 내용을 재정의 하여 사용하는 것.

- Overloading (오버로딩)

같은 method 이름을 사용하지만, 인자의 정보가 다르면 넘겨받은 정보에 맞는 method가 호출되는 것. (다양한 입력에 대해서 처리 할 수 있다.)

객체 지향 언어에서 다형성이란 하나의 클래스나 메서드가 다양한 방식으로 동작이 가능 한 것을 의미합니다. 이러하 다형성에는 오버로딩과 오버라이딩이 존재합니다.

▶ 장점: - 코드의 재활용성이 높다

           - 코딩이 절차지향보다 간편하다

           - 디버깅이 쉽다

▶ 단점: - 처리속도가 절차지향에 비해 느리다

           - 설계에 많은 시간이 소요된다.

▶ 대표적 언어: C++, JAVA, Python

### Functional Programming (함수형 프로그래밍)

Functional Programming (함수형 프로그래밍)은,

▶**변경 가능한 상태를 불변상태(Immutab)로 만들어 SideEffect를 없앤다.**

 - f(x) = y. 함수f()에 x 를 입력하면 항상 y라는 결과값이 나온다.

- 함수 안에서 상태를 관리하고 상태에 따라서 결과값이 달라지면 안된다.

- 변수 보다는 상수를 사용해 SideEffect를 차단한다.

▶ **모든 것은 Object(객체)이다.**

- 함수형 언어에서는 class외에 함수 또한 객체이다. = value(값)처럼 쓰일 수 있다.

1. 변수나 데이타에 할당 할 수 있다.
2. 객체의 인자로 넘길 수 있다.
3. 객체의 리턴값으로 리턴 할수 있다.

= 함수형 프로그래밍 언어에서는 함수가 1급 객체이다!

**+ 함수를 받아서 함수를 반환할 수 있다.**

**- Higher Order Functions (고차 함수)**

- 다른 함수를 이용해서 완전히 새로운 함수를 조립 하는 방법으로 프로그램을 만들 수 있다.

* 콜백 함수, 프로미스, 모나드 등을 사용하여 액션, 효과 또는 비동기 흐름을 추상화하거나 분리시킨다.
* 다양한 데이터 타입에 대해 동작할 수 있는 유틸리티를 만든다.
* 합성 함수나 재사용의 목적으로 커링 함수를 만들거나 인수를 함수에 부분적으로 적용한다.
* 함수 목록을 가져오고, 입력 함수의 합성을 반환한다.

**▶ 코드를 간결하게 하고 가독성을 높여 구현할 로직에 집중 시킨다.**

- boilerplate(일을 위한 일들)를 제거하여 실제 구현할 로직에만 집중 가능하고, 내부에 직접적인 함수 호출을 통해 가독성을 높일 수 있다.

**▶ 동시성 작업을 보다 쉽게 안전하게 구현한다.**

- 불변 상태의 값을 사용해, 여러 스레드에서 접근하더라도 SideEffect를 발생시키지 않는다.

- Lock, UnLock 같은 보호장치도 필요 없다.

- 데이터 변경이 불가능하기 때문에 기존 데이터의 복사본을 만들어 주는 method가 필요하다.

ex) JavaScript의 Array.map, Array.reduce

const schools = [

"Yorktown"

"Washington",

"Wakefield"

];

*// Array.map (맵핑): 모든 원소에 'High School' 문자열 추가된 새로운 배열 만들기*

const highSchools = (schools.map(school => `${school} High School`));

console.log(highSchools.join("\n"));

*// Array.reduce (축약): 배열에서 최대 값 찾기 (배열을 하나의 수로 변환)*

const result = [21, 18, 42, 40, 64, 63, 24].reduce((max, num) => num > max ? num : max, 0);

console.log(result); *// 64*

*// Array.join (합치기): 콤마(,)로 각 학교를 구분한 문자열 얻기*

console.log(schools.join(",")); *// "Yorktown", "Washington", "Wakefield"*

*// Array.filter (걸러내기): 'W'로 시작하는 학교만 있는 새로운 배열 만들기*

*// 원소를 제거하는 경우 Array.pop, Array.slice가 아닌 순수함수인 filter를 사용할 것*

console.log(schools.filter(school => school[0] === "W")); *// ["Washington", "Wakefield"]*

**▶ pure function**

* 동일한 argument(인자) 입력에 대해 항상 동일한 return(출력)값을 반환 한다.
* side-effects가 없다 (프로그래머가 바꾸고자 하는 변수 외에는 바뀌어서는 안된다.)
* 함수와 데이터를 중점으로 생각한다.

*// 순수하지 않습니다 (반환값 없이 부가 효과를 이용하고 있습니다)*

function addTaco(array) {

array.push("taco");

}

*// 순수하지 않습니다 (인자 대신 공유 변수를 이용하고 있습니다)*

function addTaco() {

return [...globalArray, "taco"];

}

*// 순수합니다*

function addTaco(array) {

return [...array, "taco"];

}

- 순수 함수는 프로그래머가 모든 것을 예측하고 통제할 수 있어야 한다!

(예측이 가능하니, 디버그가 쉬우며 테스트하는 것은 더욱 쉽다. )

- 프로그램의 의미를 변경하지 않고 결과 값으로 함수 호출을 대체할 수 있는 referentially transparent(참조 투명성) 덕분에, 함수 결과를 캐싱하여 반복적으로 사용 할 수 있는 memoization(메모이제이션) 최적화가 가능하다.

-**Function composition** (합성 함수)

→ 새로운 함수를 만들어거나 계산하기 위해 둘 이상의 함수를 조합하는 과정을 말한다. 함수형 프로그램은 여러 작은 순수 함수들로 이루어져있기 때문에 이 함수들을 연쇄적으로 또는 병렬로 호출해서 더 큰 함수를 만드는 과정으로 전체 프로그램을 구축해야 한다.

→ 메서드 체이닝 방식의 합성함수

const sum = (a, b) => a + b

const square = x => x \* x

const addTen = x => x + 10

const computeNumbers = addTen(square(sum(3, 5))) *// 74*

*// compose는 함수를 연쇄적으로 호출하면서 반환값을 전달한다*

const compose = (...fns) =>

fns.reduce((prevFn, nextFn) =>

(...args) => nextFn(prevFn(...args)),

value => value

);

*// compose의 사용*

const compute = compose(

addTen,

square,

sum

)

compute(3, 5) *// 74*

**참고**

칼럼:

<https://sung-studynote.tistory.com/109>

[선언형(declarative) vs 명령형(imperative)](https://sung-studynote.tistory.com/109" \t "_blank)

[선언형 무언가를 작업하기 위하여 어떻게 진행할 것인지를 나열하는 것을 뜻한다. 언어 : Haskell, HTML, SQL, 등... 명령형 무언가를 작업하기 위한 방법을 정의하는 것을 뜻한다. 언어 : Java, C, 등...](https://sung-studynote.tistory.com/109" \t "_blank)

[sung-studynote.tistory.com](https://sung-studynote.tistory.com/109" \t "_blank)

[**https://brownbears.tistory.com/407**](https://brownbears.tistory.com/407)

[절차지향 VS 객체지향](https://brownbears.tistory.com/407" \t "_blank)

[절차지향(Procedural Programming)이란? 절차지향 프로그래밍이란 물이 위에서 아래로 흐르는 것처럼 순차적인 처리가 중요시 되며 프로그램 전체가 유기적으로 연결되도록 만드는 프로그래밍 기법��](https://brownbears.tistory.com/407" \t "_blank)

[brownbears.tistory.com](https://brownbears.tistory.com/407" \t "_blank)

<https://lktprogrammer.tistory.com/19>

[05 [객체 지향 언어의 이해] 객체 지향의 4대 특성 04 - 다형성과 캡슐화](https://lktprogrammer.tistory.com/19" \t "_blank)

[다형성 객체 지향 언어에서 다형성이란 하나의 클래스나 메서드가 다양한 방식으로 동작이 가능 한 것을 의미합니다. 이러하 다형성에는 오버로딩과 오버라이딩이 존재합니다. 오버라이딩 오��](https://lktprogrammer.tistory.com/19" \t "_blank)

[lktprogrammer.tistory.com](https://lktprogrammer.tistory.com/19" \t "_blank)

<https://blog.ull.im/engineering/2019/04/07/functional-programming-with-javascript-in-3-steps.html>

[ull.im](https://blog.ull.im/engineering/2019/04/07/functional-programming-with-javascript-in-3-steps.html" \t "_blank)

[울려 퍼지다.  
반향하다.  
공명하다.](https://blog.ull.im/engineering/2019/04/07/functional-programming-with-javascript-in-3-steps.html" \t "_blank)

[blog.ull.im](https://blog.ull.im/engineering/2019/04/07/functional-programming-with-javascript-in-3-steps.html" \t "_blank)

<https://www.zerocho.com/category/JavaScript/post/576cafb45eb04d4c1aa35078>

[(JavaScript) 함수형 프로그래밍(Functional Programming)](https://www.zerocho.com/category/JavaScript/post/576cafb45eb04d4c1aa35078" \t "_blank)

[안녕하세요. 이번 시간에는 함수형 프로그래밍에 대해서 알아보겠습니다. 사실 함수형 프로그래밍을 중급 강좌에서 다루기엔 좀 어려운 감이 있습니다만, 다음 시간의 실습을 위해 다루도록 하](https://www.zerocho.com/category/JavaScript/post/576cafb45eb04d4c1aa35078" \t "_blank)

[www.zerocho.com](https://www.zerocho.com/category/JavaScript/post/576cafb45eb04d4c1aa35078" \t "_blank)

<https://medium.com/@lazysoul/%ED%95%A8%EC%88%98%ED%98%95-%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8D%EC%9D%B4%EB%9E%80-d881230f2a5e>

[함수형 프로그래밍이란?](https://medium.com/@lazysoul/%ED%95%A8%EC%88%98%ED%98%95-%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8D%EC%9D%B4%EB%9E%80-d881230f2a5e" \t "_blank)

[잠시 당신이 나무꾼이라고 가정해보자. 당신은 숲에서 가장 좋은 도끼를 가지고 있고, 그래서 가장 일 잘하는 나무꾼이다. 그런데 어느 날 누가 나타나서 나무를 자르는 새로운 패러다임인 전기](https://medium.com/@lazysoul/%ED%95%A8%EC%88%98%ED%98%95-%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8D%EC%9D%B4%EB%9E%80-d881230f2a5e" \t "_blank)

[medium.com](https://medium.com/@lazysoul/%ED%95%A8%EC%88%98%ED%98%95-%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8D%EC%9D%B4%EB%9E%80-d881230f2a5e" \t "_blank)

<https://velog.io/@kyusung/%ED%95%A8%EC%88%98%ED%98%95-%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8D-%EC%9A%94%EC%95%BD>

[함수형 프로그래밍 요약](https://velog.io/@kyusung/%ED%95%A8%EC%88%98%ED%98%95-%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8D-%EC%9A%94%EC%95%BD" \t "_blank)

[프로그래밍 패러다임 프로그래밍 패러다임은 프로그래머에게 프로그래밍의 관점을 갖게하고 결정하는 역할을 한다. 명령형 프로그래밍: 프로그래밍의 상태와 상태를 변경시키는 구문의 관점��](https://velog.io/@kyusung/%ED%95%A8%EC%88%98%ED%98%95-%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8D-%EC%9A%94%EC%95%BD" \t "_blank)

[velog.io](https://velog.io/@kyusung/%ED%95%A8%EC%88%98%ED%98%95-%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8D-%EC%9A%94%EC%95%BD" \t "_blank)

<http://jeonghwan-kim.github.io/js/2017/04/03/high-order-function-in-javascript.html>

[고차 함수](http://jeonghwan-kim.github.io/js/2017/04/03/high-order-function-in-javascript.html" \t "_blank)

[jeonghwan-kim.github.io](http://jeonghwan-kim.github.io/js/2017/04/03/high-order-function-in-javascript.html" \t "_blank)