코드란 무엇인가?

우리는 일반적으로 커뮤니케이션, 명료화, 비명료화라는 세 가지의 목적을 위해 코드를 사용한다.

모스부호(Morse code)는 단어를 짧고 긴 신호로 변형시킨 후 전신을 통해 의사를 전달한다. 예를 들어 my라는 단어는 송신자에 의해 "— –.—"로 암호화(encode)되고, 암호화된 소리는 수신자에 의해 다시 my라고 해독(decode)된다.

유전자 정보는 "AAAGTCTGAC"와 같이 DNA를 배열하여 암호화한다. 여기서 A는 adenine(아데닌), G는 guanine(구아닌), T는 thymine(티민), C는 cytosine(시토신)을 의미한다. 즉, 유전자 코드는 이러한 DNA 배열을 사용하여 단백질을 만들기 위한 일련의 규칙이다.

캘리포니아 건강 안전 규약은 주 입법부에서 정한 규칙들을 코드화한 법률 모음이다. 예를 들어 12504절에는 "가연성 액체는 발화점이 화씨 100도 혹은 그보다 낮은 온도인 액체를 의미한다"라고 규정하고 있다.

글쓰기가 시작된 이래 보이고 싶지 않은 메시지를 보호하기 위해 사용한 것이 코드이다. 가령 코드는 영어 알파벳의 각 문자를 숫자로 간단히 대체할 수 있다. A는 1, B는 2, C는 3...과 같은 식이다. 이런 방법으로 secret이라는 단어는 "19, 5, 3, 18, 5, 20"이 된다.

이 페이지의 그리드는 균등한 간격의 단순한 패턴에서 시작되었다. Neutral의 윤곽선 정보를 점으로 변환시키고, 이 점들을 그리드 위에 매핑하는 시스템을 코드로 정의하였다. 이 시스템은 점들의 처음 위치에서 조금씩 이동시켜 정보를 보여 주고 있다.

Morse code
1840s
모스부호는 모든 글자를 단점(dot)과 장점(dash)의 리드미컬한 배열로 부호화한다.

알고리즘

다양한 종류의 코드가 있지만 이 책에서는 주로 일련의 명령을 나타내는 코드에 대해 다룬다. 알고리즘이나 프로시저 또는 프로그램이라고 하는 이런 종류의 코드는 특정 프로세스를 상세하게 정의하여 명령대로 진행하도록 허용한다. 알고리즘이라는 단어 자체는 여러분에게 친숙하지 않을 수도 있겠지만, 의미는 잘 알고 있을 것이다. 알고리즘은 무엇을 어떻게 할 것인지를 설명하는 정확한 방법이다. 주로 컴퓨터에 무엇인가 명령을 내릴 때 사용한다. 대부분의 사람들이 목도리를 짜기 위한 패턴과 알고리즘을 연관 지어 생각하지는 않겠지만 알고 보면 이 둘은 같다.

1단 : (겉면) *겉뜨기 2회, 안뜨기 2회* 모든 단에 걸쳐 반복
2, 3, 4단 : 1단 반복
5단 : (겉면) *겉뜨기 2회, 안뜨기 2회, 교차뜨기 8회 앞* 마지막 4스티치까지 반복, 겉뜨기 2회, 안뜨기 2회
6단 : 1단 반복
1–6단을 원하는 길이까지 반복, 4단에서 마무리
겉뜨기 2회, 안뜨기 2회의 패턴으로 닫기

이처럼 어느 한 지점에서 다른 지점으로 가기 위한 방법, 가구를 조립하는 데 필요한 설명서, 그리고 이 밖의 많은 종류의 가이드라인들 역시 알고리즘이다.

Point Break의 하이킹 코스 가이드

북쪽에서 가는 방법:
- Nature Center에서부터 길을 따라간다
- Water Tower에서 우회전하여, Old Oak Tree가 보일 때까지 걸어간다
- Old Oak Tree에서 지시를 따른다

남쪽에서 가는 방법:
- Picnic Grove에서부터 Botany Trail을 따라간다
- South Meadow Trail에서 우회전한다
- Meadow Ranch Trail에서 우회전한 후 Old Oak Tree가 보일 때까지 걸어간다
- Old Oak Tree에서 지시를 따른다

Old Oak Tree에서 가는 방법:
- 나무 아래 있는 길을 따라간다
- Long Hill Trail에서 우회전한다
- Point Break에 닿을 때까지 길을 따라간다

알고리즘은 4가지 특징으로 정의할 수 있다. 이 특징들을 '하이킹 코스 가이드'와 연관 지어 생각하면 이해하기 쉬울 것이다.

첫째, 여러 가지 방법으로 알고리즘을 작성할 수 있다. A 지점에서 B 지점으로 이동하는 데에는 수많은 방법, 즉 길이 있다. 사람들마다 제각각의 방법을 선택하겠지만 그들은 모두 같은 목적지에 도착한다.

둘째, 알고리즘은 전제를 필요로 한다. 하이킹 코스 가이드는 어떻게 하이킹을 해야 하는지 알고 있다는 것을 전제하여 만들어져 있다. 어떤 신발을 신어야 하는지, 구불구불한 길은 어떻게 가야 하는지, 어느 정도의 물을 챙겨 가야 하는지 등의 방법 말이다. 이러한 사전 지식 없이 길을 나선 사람은 결국 발꿈치에 물집만 잔뜩 잡힌 채 길을 잃고 탈진할 수도 있을 것이다.

셋째, 알고리즘은 결정을 필요로 한다. 코스 가이드에 다양한 루트가 있는 경우 이용자는 이 루트들을 파악한 후 출발 지점을 선택해야 할 것이다.

넷째, 복잡한 알고리즘은 모듈로 나눈다. 코스 가이드는 보통 작은 단위로 나누어 루트를 이해하기 쉽도록 한다. 북쪽에서 출발하는 코스와 남쪽에서 출발하는 코스는 어느 지점에 이르면 만나게 되고, 두 길 모두 같은 지시를 따르게 된다.

PostScript

```
gsave
        % move to the center of the page
        306 396 translate
        % repeat from 0 to 360 in increments of 12
        0 12 360 {
                pop
                % draw line from (20,0) to (200,0)
                20 0 moveto 200 0 lineto
                % rotate 12 degrees
                12 rotate
        } for
        stroke
grestore
```

Processing

```
void setup() {
  size(800, 600);
  stroke(255, 204);
  smooth();
  background(0);
}

void draw() {
  float thickness = dist(mouseX, mouseY, pmouseX, pmouseY);
  strokeWeight(thickness);
  line(mouseX, mouseY, pmouseX, pmouseY);
}
```

PostScript
1982
포스트스크립트(PostScript)는 인쇄하기 위한 페이지를 구성할 때 사용하는 언어이다. PostScript 파일은 문서 편집기를 사용해서 작성할 수 있지만 일반적으로 GUI(Graphical User Interface)를 사용하여 만든다. GUI를 사용하면 훨씬 쉽게 파일을 만들고 편집할 수 있지만, PostScript가 갖고 있는 강력한 기능들을 잃어버리게 된다.

Processing
프로세싱(Processing)은 형태, 움직임, 인터랙션을 코딩하는 데 초점을 둔 프로그래밍 언어이자 환경으로, 자바(Java) 언어를 단순화하고 확장한 것이다. 이 프로세싱 코드를 실행하면 마우스 위치에 선을 그린다. 또한 마우스가 이동하는 속도를 반영시키면 선의 두께가 달라진다.

코드와 컴퓨터

컴퓨터 프로그래밍에서 코드(소스 코드라고 부르기도 한다)는 컴퓨터의 작동을 제어하기 위해 사용한다. 코드는 프로그래밍 언어로 작성된 알고리즘이다. 프로그래밍 언어의 종류는 수천 가지이며 매년 새로운 언어들이 개발되고 있다.
프로그래밍 언어에서 사용하는 단어와 문장 부호는 실제로 우리가 사용하는 영어와는 달라 보이지만 프로그래밍 언어에서 쓰는 코드 역시 인간이 읽고 이해하는 것을 목적으로 한다. 특히 컴퓨터 프로그래밍 언어는 어린 아이도 읽고 쓸 수 있으며, 컴퓨터에게도 아주 명확하게 지시할 수 있도록 고안되었다. 사람의 언어는 장황하고 때론 해석이 모호하며 참 많은 어휘들을 가지고 있지만 코드는 문법도 엄격하고 어휘도 적기 때문에 간결하다. 에세이나 컴퓨터 프로그램을 작성하는 코드는 모두 글을 쓰는 형식이다. 'Processing: A Programming Handbook for Visual Designers and Artist'에서 다음과 같이 설명하고 있다.

> 인간의 언어로 글을 쓸 때 저자는 단어의 모호함을 이용해서 유연성을 가진 문구를 만들어 낸다.
> 단 하나의 텍스트에서도 다양한 해석을 할 수 있으므로 저자는 자신만의 고유의 톤을 가진다. 컴퓨터 프로그램도 작자의 스타일이 있지만 모호함의 여지는 극히 적다.[1]

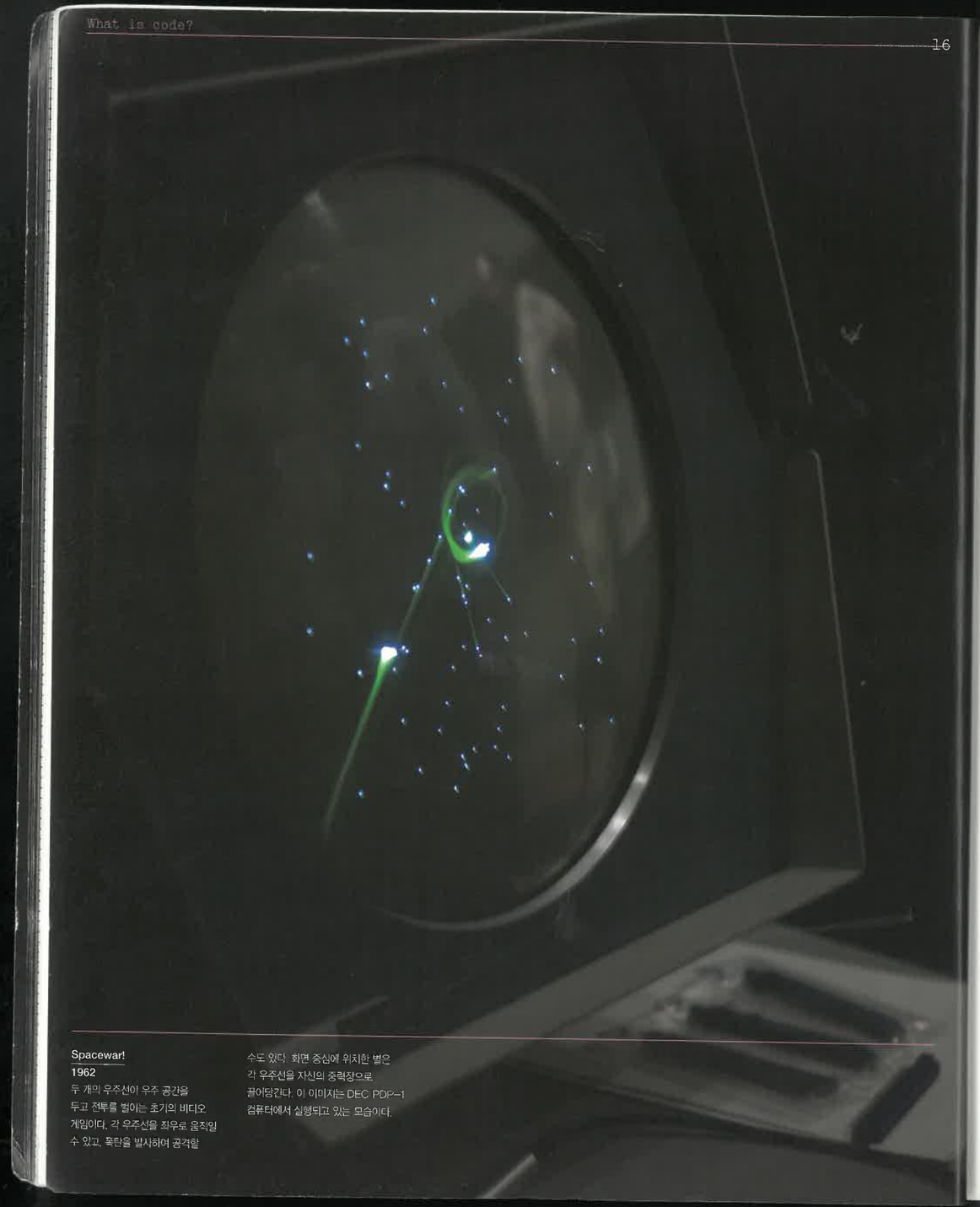
1 Casey Reas and Ben Fry, Processing: A Programming Handbook for Visual Designers and Artists (Cambridge, MA: MIT Press, 2007), 17.

사실 코드는 어떤 부분이라도 한 가지의 해석만 가능하다. 컴퓨터는 사람과 달리 의미가 정확하지 않으면 그 내용을 추측하거나 해석할 수 없다. 모든 언어에는 각각의 문법이 있지만 만약 한두 단어가 잘못 표기되어 있더라도 우리는 그 뜻을 이해할 수 있다. 하지만 컴퓨터는 그렇지 않다. 다행히도(어쩌면 불행히도) 사람들은 빠르게 적응하고 쉽게 코드의 구조를 배운다.

코드는 컴퓨터로 실행되기 전에 인간이 읽을 수 있는 포맷에서 컴퓨터가 실행할 수 있는 포맷으로 변환되어야 한다. 이러한 포맷을 보통 기계 언어(machine code), 바이너리(binaries), 실행 파일(executables)이라고 한다. 코드는 소프트웨어 내에서 변환되어 즉시 컴퓨터에서 실행할 수 있다. 기계어는 보통 1과 0을 연속적으로 사용하여 다음과 같이 표현한다.

```
0001 0001 0000 1001 0000 0001 0000
1110 0000 1001 1100 1101 0000 0101
0000 0000 1100 1001 0100 1000 0110
0101 0110 1100 0110 1100 0110 1111
0010 0001 0010 0100
```

이러한 1과 0의 나열은 소스 코드와 달라 보이지만 이것은 인간이 읽을 수 있는 코드를 문자 그대로 변환한 것이다. 변환은 컴퓨터가 해당 명령을 따를 수 있도록 하기 위해 필요한 것이다. 배열된 1과 0의 의미를 이해하는 것은 소스 코드를 읽는 것보다 훨씬 어려울 것이다. 이 기계어 포맷은 컴퓨터에 가장 근접한 명령으로, 각각의 비트(bit, 1 또는 0)는 바이트(byte, 8개 비트의 집합)들로 그룹화되어, 컴퓨터의 계산 방법과 데이터의 이동 방법을 정의한다.



Spacewar!
1962
두 개의 우주선이 우주 공간을 두고 전투를 벌이는 초기의 비디오 게임이다. 각 우주선을 좌우로 움직일 수 있고, 폭탄을 발사하여 공격할 수도 있다. 화면 중심에 위치한 별은 각 우주선을 자신의 중력장으로 끌어당긴다. 이 이미지는 DEC PDP-1 컴퓨터에서 실행되고 있는 모습이다.

코드로 생각하기

소프트웨어는 마음의 도구이다. 산업혁명이 몸을 더욱 강력하게 하기 위한 수단으로 증기 기관이나 자동차를 생산했다면, 정보혁명은 사고를 확장할 수 있는 도구를 만들었다. 우리는 소프트웨어 자산과 기술을 마음껏 이용하여 거대한 양의 정보에 접근하고 그것을 처리할 수 있게 되었다. 예를 들어 유전 정보 연구나 위키피디아는 소프트웨어의 도움 없이는 불가능했을 것이다.
소프트웨어의 사용은 이렇게 많은 양의 정보를 활용하는 능력을 높일 뿐만 아니라 새롭고 다양한 방법으로 사고할 수 있도록 한다.

절차적 표현 능력(procedural literacy)이란 용어는 이러한 잠재성을 정의하기 위해 사용되어 왔다.
산타크루스 소재 캘리포니아 대학의 컴퓨터 과학 분야 부교수인 마이클 마티스Michael Mateas는 절차적 표현 능력을 '프로세스를 읽고 쓸 수 있는 능력, 절차적 표현과 심미성을 엮는 능력'이라고 설명했다.[2] 절차적 활용 능력에는 프로그래밍이 결코 기술적인 작업만이 아니라 커뮤니케이션 행위이자 세계를 대표하는 상징적인 방법이라는 개념이 포함되어 있다.
절차적인 표현은 결코 정적이지 않으며, 오히려 가능한 형태나 움직임의 공간을 정의하는 '규칙 시스템'이다. 비디오 게임 디자이너인 이안 보고스트Ian Bogost는 이것을 그의 저서 'Persuasive Games: The Expressive Power of Videogames'에서 다음과 같이 명쾌하게 정의하고 있다.

> 절차적으로 작성한 코드는 규칙을 실행하여 어떤 표현을 생성한다. 즉, 표현 자체를 제작하는 것이 아니다. 절차적 시스템은 규칙을 따르는 모델을 기반으로 하여 행동을 생성한다. 이것은 많은 결과물을 산출해 낼 수 있는 기계이기도 하며, 각각은 하나의 동일한 지침을 따른다.[3]

'Spacewar!'와 같은 비디오 게임이 절차적 표현을 보여 주는 좋은 예다. 게임을 하기 위해서는 대적하는 두 우주선 사이의 공간과 동적 관계를 이해해야 한다. 각 플레이어는 우주선을 좌우로 움직이면서 로켓으로 공격하여 상대 우주선을 격추한다. 이 게임에서는 절차적 표현 능력을 익힌 개인이 게임 안에서의 행동들을 작고 섬세한 모듈로 분할하여 프로그램이 잘 짜여질 수 있도록 해야 했다. 게임 제작에서 가장 복잡한 것은 기술적인 것이 아니다. 구성하는 모든 요소들을 잘 연출하여 일관된 즐거운 경험을 주는 것이다. 절차적 표현 능력은 모든 프로그래밍 언어를 아우르는 일반적인 사고 방법이며, 소스 코드를 작성하는 영역 이외에서도 쓰인다.

각각의 프로그래밍 언어는 작업할 때 사고하기 위한 여러 가지 '재료'이다. 목수가 참나무, 발사나무, 소나무와 같은 다양한 목재의 고유한 속성을 알 듯, 소프트웨어를 잘 아는 사람은 각 프로그래밍 언어의 고유한 성격을 알고 있다. 책상을 만드는 목수는 비용, 내구성, 심미성 등을 고려하여 나무를 고르지만, 프로그래머는 예산, 운영체제, 심미성을 고려하여 프로그래밍 언어를 선택한다.[4] 각 프로그래밍 언어의 구문과 문법은 해당 언어 내에서 가능한 것들을 구조화한다. 그렇기 때문에 프로그래머들은 해당 언어가 가진 어포던스(또는 행동 가능성)와 제약을 고려하여 작업을 구상한다.

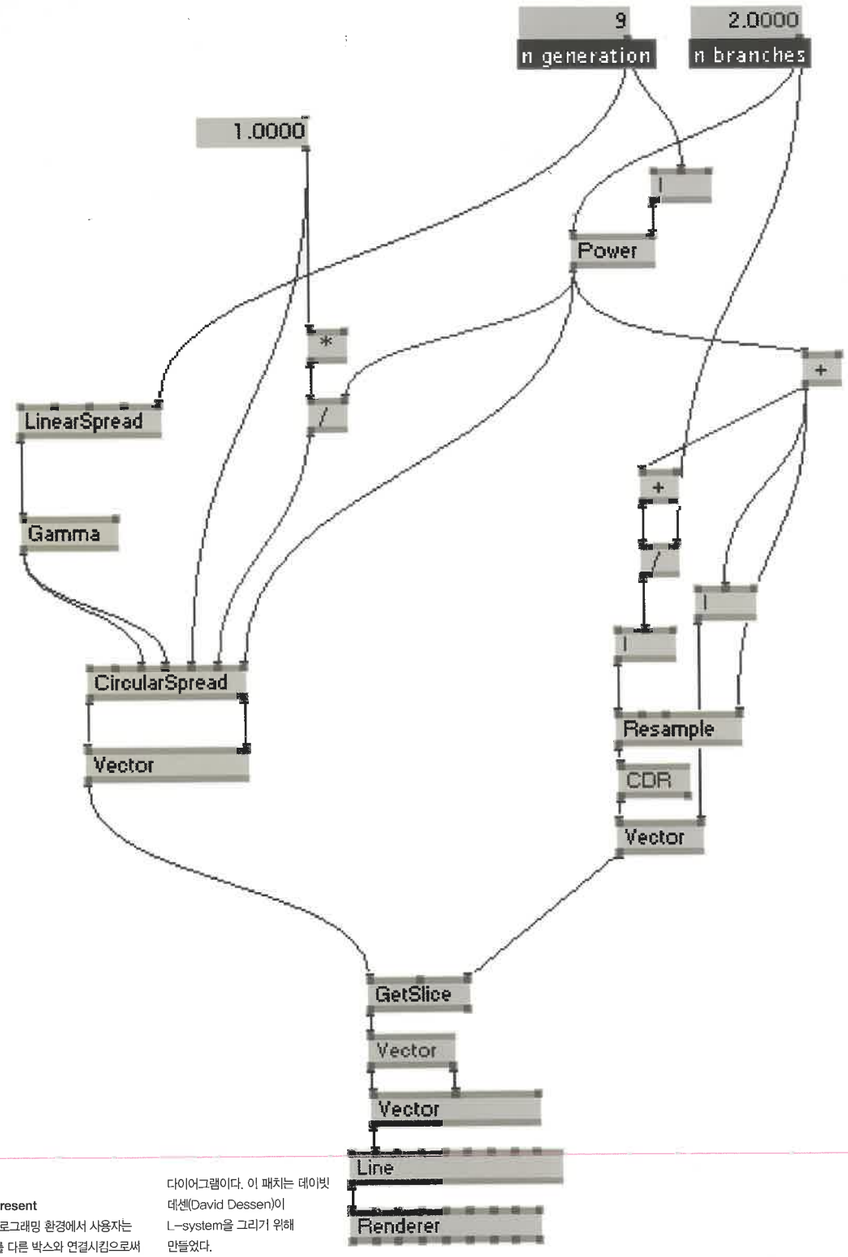
프로그래밍 언어는 어떤 특정한 모드로 사고하도록 유도한다. 그것은 서로 다른 언어인 BASIC과 LOGO를 비교하면 명확하게 알 수 있다. 두 프로그래밍 환경에서 삼각형을 그리려면 공간에 대한 다른 접근과 이해가 필요하다. BASIC은 기본적으로 좌표 시스템을 가지고 있으므로 좌표에 대해서 알아야 한다. 예를 들어 선은 하나의 좌표와 다른 좌표를 연결시켜 그린다.[5]

2 Michael Mateas, "Procedural Literacy: Educating the New Media Practitioner," Beyond Fun, ed. Drew Davidson (Pittsburgh, PA: ETC Press, 2008), 67.

3 Ian Bogost, Persuasive Games: The Expressive Power of Videogames (Cambridge,MA: MIT Press, 2007), 4.

4 특정 프로그래밍 언어를 선호하는 데에는 여러 가지 이유가 있다. 예를 들어 쉽게 코딩할 수 있는 언어는 대부분 실행 속도를 희생한다. 비교적 잘 쓰이지 않는 언어는 프로그래머들이 그 언어를 알 수 있는 기회가 줄어든다.

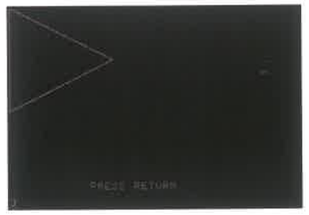
5 BASIC 언어는 1964년에 프로그래밍을 전공하지 않은 학생들을 가르치기 위해서 처음 개발되었다. 개인용 컴퓨터 시대의 초기에는 이를 기반으로 한 언어들이 주로 어린이들이나 프로그래밍을 취미로 즐기는 사람들을 가르치기 위해 사용되었다.



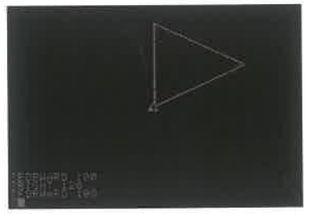
vvvv
1998-present
vvvv 프로그래밍 환경에서 사용자는 각 박스를 다른 박스와 연결시킴으로써 데이터 흐름을 제어한다. 이 프로그램은 노드를 연결하여 표현한 비주얼 다이어그램이다. 이 패치는 데이빗 데센(David Dessen)이 L-system을 그리기 위해 만들었다.

반면 LOGO는 기하학을 아직 배우지 않은 어린 아이들을 위해 개발된 언어로, 사용자가 좌우의 차이와 각도에 대해 이해만 한다면 형태를 코딩할 수 있다. LOGO에서는 화면상에서 거북이의 이동 경로를 지시하여 선을 그려 나간다. 아이들은 자신이 거북이라고 상상하고 앞으로 전진, 우회전, 전진, 다시 우회전 그리고 전진하여 삼각형을 완성한다.[6] 비록 두 언어가 그리는 모양은 같지만 BASIC은 객관성을, LOGO는 탐구성을 기르도록 한다. 또한 LOGO는 프로그래머가 마음 속으로 코드를 실행해 볼 수 있도록 하므로 절차적 표현 능력을 기르는 데 도움이 된다.

6 1967년에 처음 개발된 LOGO는 화면상에서 삼각형 모양의 거북이를 움직여서 선을 그린다. 이 언어의 초기 버전에서는 로봇 거북이가 명령에 따라 방의 이곳저곳을 이동하며 그림을 그렸다.



BASIC
10 HGR : HCOLOR = 3
20 HPLOT 0, 0 TO 100, 50 TO 0, 100 TO 0, 0
30 END



LOGO
FORWARD 100
RIGHT 120
FORWARD 100
RIGHT 120
FORWARD 100

비주얼 프로그래밍 언어(그래픽 프로그래밍 언어라고 부르기도 한다)는 코드로 생각하는 또 다른 방법을 제공한다. 비주얼 프로그래밍 언어로 프로그램을 작성하는 것은 글로 쓰는 것이 아니라 다이어그램을 그리는 것과 비슷하다. 예술의 영역에서 인기 있는 비주얼 프로그래밍 언어는 Max, Pure Data, vvvv이다. 이들은 아날로그 신시사이저에 연결된 패치 케이블을 사용하여 사운드를 구성하는 방식에서 영향을 받았다. 화면상에서 가상의 케이블들은 선으로 대체되었으며, 이 선들은 프로그래밍 모듈들을 서로 연결하면서 소프트웨어 역할을 규정한다. 비주얼 프로그래밍 언어는 이미지와 사운드를 생성하고 필터링하는 것은 쉽지만, 길고 복잡한 프로그램을 작성할 때는 오히려 다루기가 어렵다. 사실 Max 자체는 비주얼 프로그래밍이 아닌 텍스트 프로그래밍 언어인 C++로 쓰였다.

PROPOSAL FOR WALL DRAWING, INFORMATION SHOW

Within four adjacent squares,
each 4' by 4',
four draftsmen will be employed
at $4.00/hour
for four hours a day
and for four days to draw straight lines
4 inches long
using four different colored pencils;
9H black, red, yellow and blue.
Each draftsmen will use the same color throughout
the four day period,
working on a different square each day.

Proposal for a Wall Drawing, Information Show
Sol LeWitt, 1970
솔 르윗(Sol LeWitt)은 드로잉을 하기 위한 지시문을 만들어 제도사가 직접 제작할 수 있도록 했다. 숙련된 제도사들은 이 지시문을 따라 드로잉을 진행한다.

코드와 예술

7 Seymour Papert, The Children's Machine: Rethinking School in the Age of the Computer (New York: Basic Books, 1994), 157.

8 Jack Burnham, Software—Information Technology: Its New Meaning for Art (New York: The Jewish Museum, 1970), 10.

9 The New Media Reader (Cambridge, MA: MIT Press, 2003), 255.

코드는 1940년대 초에 과학과 공학 분야의 연구를 지원하기 위해 개발되었다. 컴퓨터와 창의력 연구에서 선구적 역할을 한 세이머 페퍼트Seymour Papert는 당시 상황을 다음과 같이 설명하고 있다.

> 세계는 전쟁 중이었다. 수학자는 시간에 쫓기며 복잡한 계산을 해내야만 했다. 무기의 설계 및 사용에 필요한 수적 계산, 최대한 빠른 시간 안에 정보를 파악하기 위해 복잡한 코드들을 해독해내기 위한 논리적인 조작.... 당시의 수학자들은 보다 부드러운 표현 방법으로 사용자 친화적인 컴퓨터를 만든다는 것은 생각조차 하지 않을 것 같았다.[7]

소프트웨어와 예술이 통합되기까지는 시간이 걸렸다. 안타깝게도 예술에서 사용하는 많은 언어들이 처음부터 예술 영역에서 표현하기 위해 설계된 것은 아니었다. 일반적으로 디자이너, 건축가, 예술가들이 원하는 소프트웨어는 과학자, 수학자, 공학자들이 생각하는 것과는 다르다. 현재 가장 많이 사용하고 있는 C++이나 Java와 같은 언어로 시각적 형태를 만들기 위해 필요한 기술을 습득하기까지는 보통 몇 년이 걸린다.

1950년대부터 1960년대까지 예술가들이 소프트웨어 또는 비물질화 및 시스템 미학과 같은 소프트웨어와 관련된 주제로 실험한 작업들을 발표하면서 코드에 대한 새로운 생각들이 모습을 드러냈다. 이러한 시도들은 1968년, 런던의 현대 미술 학회Institute of Contemporary Arts에서 열린 'Cybernetic Serendipity' 전에서 처음으로 대중에게 소개되었다. 1970년에는 뉴욕의 유태인 박물관Jewish Museum에서 열린 'Software — Information Technology: Its New Meaning for Art' 전과 뉴욕 현대미술관MoMA에서 열린 'Information' 전에서 소개되었다. 'Software' 전의 큐레이터인 잭 번햄Jack Burnham은 출품된 작품들에 대해 다음과 같이 설명했다. "예술은 상호 작용이며, 소통과 에너지 교환의 근본적인 구조들을 다룬다"[8] 한스 하케Hans Haacke의 야심작인 'Visitor's Profile'은 예술계에 대한 비판적 표현으로서 미술관 방문자들의 특권적인 사회적 지위를 파헤치고자 했다. 컴퓨터 인터페이스를 사용하여 방문자가 답변한 개인 정보를 표로 정리했다. 레스 레빈Les Levine은 소프트웨어의 맥락에서 논의되는 사진 작품 'Systems Burn-off X Residual Software'를 전시했다. 레빈은 이미지는 하드웨어이며 이미지에 대한 정보는 소프트웨어라고 주장하면서 다음과 같은 도발적인 표현도 서슴지 않았다. "물질이나 사물과 아무런 관련이 없는 모든 행위는 소프트웨어의 결과물이다."[9] 레빈의 작품뿐만 아니라 뉴욕 현대미술관의 'Information' 전에 소개된 대부분의 작품들은 개념 예술conceptual art의 특징을 보여 준다.

CLOUD PIECE
Imagine the clouds dripping.
Dig a hole in your garden to put them in.

1963 Spring

같은 시대에, 멜 보크너Mel Bochner, 존 케이지John Cage, 앨런 캐프로Allan Kaprow, 솔 르윗Sol LeWitt, 요코 오노Yoko Ono, 그리고 라 몬테 영La Monte Young과 같은 많은 예술가와 음악가는 지시문을 작성하거나 다이어그램을 그림으로써 새로운 방식의 개념과 프로세스를 기반으로 한 예술을 만들어 냈다. 예를 들어 르윗은 손으로 그림을 그리는 대신 자신의 생각들을 '지시문'의 형태로 코드화하였고 이것을 사용하여 그림을 그렸다. 그는 일련의 규칙을 작성하여 제도사가 이에 따라 작업을 수행하도록 했지만, 그 규칙에 대해서는 자유롭게 해석이 가능하기 때문에 여러 가지 다양한 결과들이 나올 수 있었다. 오노의 'Cloud Piece'와 같은 작품은 삶에 대한 지시문이다. 그녀는 짤막한 글로 독자들에게 웃거나, 그리거나, 앉거나 또는 날아다닐 것을 요구한다. 마치 프로그래머처럼 예술가들은 사람들에게 어떠한 행동을 하도록 지시한다. 영어를 프로그래밍 언어로 사용하여 모호함이나 해석, 그리고 심지어는 모순까지도 보여 주었다.

이러한 개념 예술의 탐구가 이루어지는 동안 공학자들은 시각적 이미지를 생성하기 위한 프로그래밍 시스템을 개발하고 있었다. 케네스 놀턴Kenneth C. Knowlton은 1963년에 벨 연구소에서 애니메이션 제작에 특화된 BEFLIX 프로그래밍 언어를 만들었다. 그리고 그는 이 언어를 사용하여 예술가인 스탠 밴더빅Stan VanDerBeek,

Electronic Numerical Integrator And Computer (ENIAC)
1943-46
초기의 디지털 컴퓨터는 지금의 컴퓨터와 매우 달랐다. ENIAC의 가격은 약 50만 달러였으며 무게 또한 30톤 이상 나갔다. 이 사진은 두 컴퓨터 프로그래머들의 모습을 보여 주고 있다.

Cloud Piece
Yoko Ono, 1963
요코 오노(Yoko Ono)의 작업은 지시문 형태이다. 이 글을 읽고 사람들은 상상하거나 행동을 취한다.

BASIC

```
10 DEFINT A-Z' DRAW100SQ
20 CLS
30 MOVE$="!AX"
40 DRW$="!AY"
50 OPEN "COM1:1200,0,7,1" AS #1
60 PRINT #1, "!AE";
70 FOR ROW=0 TO 90 STEP 10
80  FOR CLM=0 TO 90 STEP 10
90       GOSUB 310
100      PRINT #1, MOVE$+STR$(ROW)+STR$(CLM)
110      GOSUB 310
120      PRINT #1, DRW$+STR$(ROW+10)+STR$(CLM)
130      GOSUB 310
140      PRINT #1, DRW$+STR$(ROW+10)+STR$(CLM+10)
150      GOSUB 310
160      PRINT #1, DRW$+STR$(ROW)+STR$(CLM+10)
170      GOSUB 310
180      PRINT #1, DRW$+STR$(ROW)+STR$(CLM)+";"
190 NEXT CLM
200 NEXT ROW
210 PRINT
220 GOTO 70
230 '
240 '
250 '
260 '
270 'XON/XOFF subroutine
```

Hypertalk

```
on mouseUp
  put "100,100" into pos
  repeat with x = 1 to the number of card buttons
    set the location of card button x to pos
    add 15 to item 1 of pos
  end repeat
end mouseUp
```

BASIC
1964
이 프로그램은 100개의 사각형으로 된 그리드를 그린다. BASIC 언어를 사용하여 플로터라고 일려진 드로잉 장치를 제어하는 것을 보여 준다.

HyperTalk
1987
HyperTalk 언어는 다른 프로그래밍 언어들에 비해 영어와 조금 더 비슷한 모습으로, 초보자들도 쉽게 배울 수 있다.

릴리안 슈왈츠Lillian F. Schwartz와 함께 초기의 컴퓨터 영화를 제작했다.
존 휘트니John Whitney Sr.는 1966년에 IBM의 잭 사이트론Dr. Jack Citron이 개발한 프로그래밍 라이브러리인 GRAF를 사용하여 영화 'Permutations'를 제작했다.

BEFLIX와 GRAF는 모두 FORTRAN 언어를 바탕으로 만들어진 것이다. 이런 초기의 연구들을 시작으로 예술이라는 분명한 목적을 위한 프로그래밍 언어의 발전이 점차 기세를 더해 오늘날의 고조된 모습으로 이어졌다.

1980년대에 개인용 컴퓨터가 급격히 보급되면서 더 많은 사람들이 프로그래밍을 접할 수 있게 되었고, 자연스럽게 HyperTalk의 개발로 이어졌다. HyperTalk는 Apple의 응용프로그램인 HyperCard(초기의 하이퍼미디어 시스템)를 위한 프로그래밍 언어이다. 같은 종류의 언어인 Lingo는 1988년 Adobe Director(이전에는 Macromedia Director 그리고 그 이전에는 MacroMind Director였다)의 처음 출시에 맞춰 개발되었다.

Lingo는 1990년대 초, World Wide Web이 등장할 때까지 많은 디자이너와 예술가들이 사용했던 최초의 프로그래밍 언어였다. 웹의 초창기에는 그래픽 프로그래밍 연구에 집중했는데 주로 ActionScript 언어가 사용되었다. 오늘날에는 예술과 건축 분야에서의 프로그래밍 활용 능력이 향상되면서 선택할 수 있는 프로그래밍이 늘어나고 있으며, 이 책에서도 이와 관련된 많은 부분들을 다루고 있다.

코드는 컴퓨터 화면 속에서만이 아니라 물리적 공간에도 영향을 주고 있다. 제품, 건축, 그리고 설치 분야에서 어떤 부분들을 제작하는 데에도 코드가 사용되고 있다. 코드를 사용하여 인쇄물로 출력되는 파일을 만들거나 컴퓨터로 제어하는 기계에서 목재, 금속, 플라스틱 등의 재료를 절단하고 조립하기도 한다. 코드는 어느새 화면 밖으로 나와 모든 물리적인 세계를 제어하기 시작했다. 이에 대한 예시들을 "형태와 컴퓨터"의 '형태 만들기(37쪽)' 부분과 이 책의 곳곳에서 소개하고 있다.