

CS50`S Introduction to Computer Science-

HARVARD UNIVERSITY

Scratch

LECTURER in English : DAVID MALAN

Transcription in to Bangal : MU. MAHFUZ YASIN

DAVID MALAN:

আচ্ছা, শুরু করা যাক।

এটা হলো CS50, হার্ভার্ড বিশ্ববিদ্যালয়ের কম্পিউটার বিজ্ঞানের এবং প্রোগ্রামিংয়ের শিল্পকৌশলের প্রতি একটি বুদ্ধিবৃত্তিক অনুশীলনের পরিচিতি।

আমার নাম ডেভিড মালান, এবং আমি নিজেই কিছু বছর আগে এই কোর্সটি নিয়েছিলাম— যদিও আমি প্রায় নেই-ইনি।
আমার কলেজের প্রথম বছর ছিল সেটা, আমরা তখন ম্যাথিউস হলে থাকতাম— যাঁরা জানেন, [উল্লাসধ্বনি] হ্যাঁ, ম্যাথিউস!
আমাদের গর্ব ছিল এই যে, সেই কক্ষে মাত্র তিন বছর আগেই ছিলেন ম্যাট ডেমন।

কিন্তু, প্রথম বছরে আমার আসলে সাহস হয়নি এই শ্রেণিকক্ষে পা রাখার— এমনকি কম্পিউটার সায়েন্স শেখারও না।
কম্পিউটার নিয়ে আমি মোটামুটি স্বাচ্ছন্দ্যবোধ করতাম, তবে অন্য যারা খুবই দক্ষ ছিল, তাদের মধ্যে ছিলাম না।

ফলে প্রথম বছরে আমি কম্পিউটার সায়েন্স থেকে দূরে থাকি।
তার বদলে আমি অনেকগুলো গভর্নমেন্ট (সরকার) বিষয়ক কোর্স করি।
হাইস্কুলে ইতিহাস আমার ভালো লাগত। বিশেষ করে আমার সিনিয়র বছরে নেওয়া কনস্টিটিউশনাল ল' কোর্সটি ছিল দারুণ।
তাই আমি ভেবেছিলাম— যেহেতু এটা আমার স্বাচ্ছন্দ্যের জায়গা, কলেজেও এটিই করা উচিত।

তাই আমার প্রথম এক-দেড় বছর আমি গভর্নমেন্ট মেজর হিসেবে কাটাই।
কিন্তু সেকেন্ড ইয়ারে, আমি তখন ম্যাথার হাউসে থাকি (কেউ ম্যাথার থেকে আছেন?),
ক্লাসের প্রথম সপ্তাহে আমি কিছু বন্ধুকে অনুসরণ করি এক ক্লাসে— নাম ছিল CS50।

সত্যি বলতে, আমি ক্লাসে পা রাখতেই সেই সময়কার অধ্যাপক, প্রখ্যাত কম্পিউটার বিজ্ঞানী ব্রায়ান কার্নিহান (বর্তমানে প্রিন্সটনে)—
তঁার লেকচার শুনেই আমি মুগ্ধ হয়ে যাই।
তারপর শুরুর সপ্তাহ, যখন অ্যাসাইনমেন্ট রিলিজ হতো, আমি বসে যেতাম কাজ করতে।

আমরা অবশ্য এটি সুপারিশ করি না যে শুরুর রাত ৮টায় হোমওয়ার্ক শুরু করতে হবে।
তবে এটা আমার কাছে ছিল এক ধরনের ইজিত— আমি যেন নিজের স্থান খুঁজে পেয়েছি।

সব শিক্ষার্থীর জন্য এমনটা হবে না, অবশ্যই।
আমাদের কোনো প্রত্যাশা নেই যে আপনি এই একটি কম্পিউটার সায়েন্স কোর্স করার পর অন্য কোর্সও নিতে বাধ্য হবেন বা চাইবেন।

তবে **Computer Science** (কম্পিউটার সায়েন্স) এবং বিশেষ করে **CS50**-এর সবচেয়ে শক্তিশালী দিক হলো—
এটি এতটাই ব্যবহারিক (applicable) যে আপনি **arts** (কলা), **humanities** (মানবিক শাস্ত্র), **social sciences** (সমাজবিজ্ঞান),

natural sciences (প্রাকৃতিক বিজ্ঞান)— যেটাই পড়েন না কেন,

এই কোর্স থেকে পাওয়া **concepts** (ধারণাসমূহ) এবং **programming skills** (প্রোগ্রামিং দক্ষতা) সবখানেই কাজে লাগবে।

এটা অবশ্যই চ্যালেঞ্জিং (challenging) হবে, এবং আমিও যখন এটি করেছিলাম, কঠিন লেগেছিল।

এই যে দেখুন, এখানে একটি বিখ্যাত **MIT hack** (এমআইটি-র কৌশল) এর ছবি আছে—

এটা বোঝায় যে MIT-তে পড়াশোনা করা নাকি **drinking from a fire hose** (ফায়ার হোজ থেকে পানি খাওয়ার মতো)।

মানে হলো— এত বেশি তথ্য, এত নতুন জিনিস একসঙ্গে আসবে,

প্রতিদিন সবকিছু একবারেই শিখে ফেলা সম্ভব হবে না।

তবে সেই চ্যালেঞ্জ মোকাবেলা করার মধ্য দিয়েই, কোর্স শেষে আপনি অনেক বেশি লাভবান হবেন।

আপনার শুধু কম্পিউটার সায়েন্স ও প্রোগ্রামিং-ই নয়,

বরং **new technologies** (নতুন প্রযুক্তি) নিজে নিজে শেখার সক্ষমতাও তৈরি হবে।

পরবর্তী তিন মাসেরও বেশি সময়জুড়ে থাকবে **Teaching Fellows, Teaching Assistants, Course Assistants**, এবং আমিও থাকবো—

আপনাদের **guide** (নির্দেশনা দিতে) করার জন্য।

কিন্তু সেমিস্টারের শেষ দিকে আমরা চাই আপনি **training wheels off** (সহজীকরণ সরঞ্জাম ছাড়াই) চলতে শিখুন—

অর্থাৎ আপনি নিজেই নতুন জিনিস শেখার উপযুক্ত হয়ে উঠবেন।

Take comfort (স্বস্তি পান) এ জেনে যে, বেশিরভাগ CS50 শিক্ষার্থী এর আগে কখনোই **CS course** (কম্পিউটার সায়েন্স কোর্স) নেয়নি।

Syllabus (সিলেবাস)-এও বলা আছে— এই কোর্সে মূলত গুরুত্বপূর্ণ নয় যে আপনি আপনার ক্লাসমেটদের তুলনায় কোথায় আছেন, বরং আপনি নিজের শুরুর থেকে কোথায় পৌঁছেছেন, সেটিই গুরুত্বপূর্ণ।

আর শুরু মানেই আজ।

তাই ভেবে দেখুন আপনি বর্তমানে **comfortable or uncomfortable** (স্বাচ্ছন্দ্যবোধ করছেন না কি করছেন না)

computing, computer science, বা **programming** নিয়ে,

বিশেষ করে যদি আপনি এগুলো আগে কখনো ক্লাসে শেখেননি।

তিন মাস পরের আপনি, আজকের আপনার তুলনায় কতদূর এগিয়েছেন— এই **difference** বা **delta** (পার্থক্য) ই হবে আসল মূল্যায়ন।

এবং অন্যদের— আপনার ডানে, বামে, সামনে বা পেছনে কে কত জানে, সেটা একেবারেই গুরুত্বহীন।

এখন আমি একটু অনুপ্রেরণা দিতে চাই, যদি অনুমতি দেন।

এই যে ছবিটা দেখছেন— এটা আমার নিজের **first homework assignment** (প্রথম হোমওয়ার্ক অ্যাসাইনমেন্ট) CS50 থেকে, ১৯৯৬ সালের।

দয়া করে খেয়াল করুন, এটি ছিল একটি **"Hello, World" program** (হ্যালো ওয়ার্ল্ড প্রোগ্রাম),

যেটা আমরা নিজেরাও আগামী সপ্তাহে লিখব।

এটা মূলত সবচেয়ে ছোট, সহজ প্রোগ্রাম যা আপনি **C programming language** (সি প্রোগ্রামিং ভাষা)-এ লিখতে পারেন।

তবুও, আমি কিভাবে যেন এতে **minus 2** (মাইনাস দুই) পেয়েছিলাম!

মানে, আমরা সবাই ভুল করব শেখার পথে।

কিন্তু লক্ষ্য হলো শেখা এবং সেই প্রক্রিয়াটি উপভোগ করা।

অবশেষে, আমার মতোই আপনি কোর্স শেষে গর্ব করে পরবেন

"I took CS50" T-shirt (CS50 কোর্স সম্পন্ন করেছি এমন টি-শার্ট),

যেটা আমাদের এক **tradition** (ঐতিহ্য)।

কিন্তু এটিই একমাত্র tradition নয়।

CS50-তে রয়েছে আরও অনেক tradition,
চাকরির মতো পরিপাটি না, বরং আনন্দদায়ক, যেমন **Puzzle Day** (ধাঁধার দিন)।

মাত্র কিছুদিন পরেই আমরা শুরু করব **CS50 Puzzle Day**—
যেটা এক দারুণ সুযোগ বন্ধুদের সঙ্গে **pizza, prizes**, আর **logic puzzles** (যুক্তিভিত্তিক ধাঁধা) উপভোগ করার।

এবং এই অনুষ্ঠানের মূল উদ্দেশ্য হলো এটি বোঝানো যে—
Computer Science মানে কেবল **programming, C**, বা **Python** নয়।

এর আসল বিষয় হলো **problem solving** (সমস্যা সমাধান),
বিশেষ করে অন্য **smart people** (মেধাবী লোকজন)-এর সাথে **collaboration** (সহযোগিতা) করে।

সেমিস্টারের শেষের দিকে আপনি অংশ নেবেন **CS50 Hackathon**-এ—
একটি **overnight event** (রাতভর ইভেন্ট), যেখানে আপনি কাজ করবেন নিজের **final project**-এ।

এরপর হবে **CS50 Fair**,
যেটা এক ধরনের **exhibition** (প্রদর্শনী) হবে পুরো ক্যাম্পাসের জন্য—
students, faculty, এবং **staff** সবাই আপনার **final project** দেখতে পারবে।

আপনার **project** হতে পারে একটি **web app, mobile app**, অথবা অন্য যেকোনো কিছু
যা আপনি তৈরি করবেন **term's end** (সেমিস্টারের শেষে)।

এবং সত্যি বলতে, আমরা চাই আপনার প্রজেক্ট হোক এমন কিছু
যা আমরা আপনাকে শিখাইনি!

কারণ সেটাই হবে প্রমাণ— আপনি সত্যিই **ready** (প্রস্তুত)।

এখন, আপনাদের **CS50's past** (অতীতের CS50) দেখাতে চাই
একটি **short video** (ছোট ভিডিও)-র মাধ্যমে।
Lights dim (আলো কমিয়ে দিন) করলে ভালো হয়।

ভিডিও চলবে...

♪ [VIDEO PLAYBACK] ♪
♪ [MUSIC PLAYING] ♪

DAVID MALAN:

আচ্ছা, তাহলে CS50-তে এবং কম্পিউটার সায়েন্সে আপনাকে স্বাগতম।
তাহলে, **Computer Science** (কম্পিউটার সায়েন্স) কী?

সহজভাবে বললে, এটি হলো **the study of information**—
অর্থাৎ, তথ্য কীভাবে উপস্থাপন (**represent**) ও প্রক্রিয়াকরণ (**process**) করা যায়।

কিন্তু আরও গভীরে গেলে, এই কোর্সে আমরা যেটা শেখাবো তা হলো—
Computational Thinking (কম্পিউটেশনাল চিন্তাভাবনা)।
মানে, কম্পিউটার সায়েন্সের ধারণাগুলোকে বাস্তব সমস্যার ওপর প্রয়োগ করা।

এই কোর্সের পর, আপনি ক্লাসের ভেতরে বা বাইরে, নিজের আগ্রহের বিষয় নিয়ে
Problem Solving (সমস্যা সমাধান)-এ এটি প্রয়োগ করতে পারবেন।

সুতরাং, **Computer Science** হলো আসলে **Problem Solving**।
এবং এই জন্যই এর **global applicability** (বিশ্বব্যাপী প্রাসঙ্গিকতা) রয়েছে।

এখন, **Problem Solving** মানে কী?

চলুন একটি **mental image** (মনের ছবি) দেখি।

একটি **problem** আছে— মানে কোনো একটা **input** (ইনপুট) যেটি আপনি **solve** করতে চান।
আপনার **goal** হলো একটি **solution** (সমাধান) তৈরি করা, যা হবে **output** (আউটপুট)।
তাহলে, আপনি ইনপুটকে কীভাবে প্রসেস করে আউটপুটে রূপান্তর করছেন— সেটিই মূল জায়গা।

কিন্তু, তার আগে আমাদের **agree** করতে হবে—

এই ইনপুট ও আউটপুটগুলোকে কীভাবে **represent** করবো?

বিশ্বব্যাপী **standardized** পদ্ধতিতে, যাতে করে এগুলো **computers** (কম্পিউটার) বুঝতে পারে—
হোক সেটা **laptop, desktop, mobile phone**, বা অন্য কোনো **electronic device**।

তাহলে আমরা কীভাবে তথ্য উপস্থাপন করতে পারি?

অনেকভাবে পারি। উদাহরণস্বরূপ, যদি আমি ছোট কোনো কক্ষে **attendance** (উপস্থিতি) নেই,
তবে আমি **human hand** দিয়ে 1, 2, 3, 4, 5... এভাবে কাউন্ট করতে পারি।

এটি একধরনের **Unary Notation** (ইউনারি পদ্ধতি), যেটি গাণিতিকভাবে পরিচিত **Base 1** নামে।
কারণ আপনি একবারে একটি করে সংখ্যা যোগ করছেন।

Quick question— এক হাতে আপনি সর্বোচ্চ কত পর্যন্ত গুনতে পারেন?

Five (পাঁচ)?

না, যদি আপনি **different base system** (ভিন্ন ভিত্তি পদ্ধতি) ব্যবহার করেন, তাহলে আরও বেশি।

আসলে, আমি এমন প্যাটার্ন তৈরি করতে পারি যা দিয়ে এক হাতে **31** পর্যন্ত গুনতে পারি।

কিন্তু কীভাবে?

এবার আমরা **Base 2** অর্থাৎ **Binary System** (দ্বিমিক সংখ্যা পদ্ধতি)-তে চলে আসি।

ধরি:

- একটি আঙুল নিচে = 0
- একটি আঙুল ওপরে = 1

এভাবে প্রতিটি আঙুলের **two possible states** (দুটি অবস্থা) আছে—

Down or Up, যাকে বলা হয় 0 বা 1।

এই পদ্ধতিতে এক হাতে আপনি $2^5 = 32$ **possibilities** পান (0 থেকে 31 পর্যন্ত)।

এই হল **Binary** (দ্বিমিক), যা কম্পিউটার বোঝে।

কম্পিউটার কেবল **Zeros and Ones** (০ ও ১) বোঝে।

এগুলোকে বলা হয় **Binary Digits**, সংক্ষেপে **Bits**।

Bit মানে হলো— একটি **Binary Digit**, অর্থাৎ 0 বা 1।

এখন আমরা যদি **information** (তথ্য) উপস্থাপন করতে চাই **computers** দিয়ে—

আমরা কীভাবে **zero** এবং **one** ব্যবহার করবো?

ধরি, আমি একটা **light bulb** (বাতি)-এর কথা ভাবছি।

যদি **switch off** থাকে, তাহলে এটা **0** বোঝায়।

যদি **switch on** থাকে, তাহলে এটা **1** বোঝায়।

আপনার **computer** বা **phone**-এ রয়েছে কোটি কোটি **transistors**—

যেগুলো ছোট ছোট **electronic switches**।

প্রত্যেকটি **transistor** হয় **on** না হয় **off** হয়।

এবং এভাবেই **data** সংরক্ষণ করা হয়—

0 মানে **off**, আর **1** মানে **on**।

তবে প্রশ্ন হলো—

যদি কেবল **1 bit** থাকে, তাহলে তো আমরা কেবল **0** এবং **1** গুনতে পারি।

তাহলে আমরা যদি বড় সংখ্যা গুনতে চাই, তাহলে কী করবো?

আমরা **more bits** (আরও বিট) ব্যবহার করবো।

ধরি আমাদের সামনে আছে **3 bits**— মানে **3 light bulbs**।

তাহলে আমরা এইভাবে সংখ্যা উপস্থাপন করতে পারি:

- **000** → 0
- **001** → 1
- **010** → 2
- **011** → 3
- **100** → 4
- **101** → 5
- **110** → 6
- **111** → 7

এখানে **8 unique patterns** পাওয়া যাচ্ছে।

তাই ৩টি বিট দিয়ে আমরা গুনতে পারি **0 থেকে 7**, মোট **8 possibilities**।

কিন্তু কেন **7** পর্যন্ত?

কারণ, গণনা **zero** থেকে শুরু হয়েছে, **one** থেকে নয়।

এখন যদি আমরা আরও বড় সংখ্যা গুনতে চাই—

ধরি **8 পর্যন্ত**, তাহলে আমাদের দরকার **4 bits**।

- **1000 (Binary) = 8 (Decimal)**

এবং এভাবে আরও যেতে যেতে বড় বড় সংখ্যা গোনা যায়।

কিন্তু **1 bit, 3 bits**, এমনকি **4 bits** ব্যবহারিকভাবে যথেষ্ট নয়।

তাই সাধারণত ব্যবহার করা হয় **1 Byte**।

How many bits in a Byte?

→ **8 bits**।

কেন ৮? কারণ এটা **power of 2**, এবং **electronically convenient**।

তাহলে ৮ বিট দিয়ে আমরা সর্বোচ্চ গুনতে পারি:

- $2^8 = 256$ possibilities (0 থেকে 255 পর্যন্ত)

তাহলে **maximum number** যা ৮ বিট দিয়ে উপস্থাপন করা যায় তা হলো **255**।

এই জন্য আমরা অনেক সময় **computer systems**-এ দেখি সংখ্যাগুলো **256, 255, 128** ইত্যাদি, কারণ সেগুলো **power of 2**।

এভাবে কম্পিউটার **Binary** ব্যবস্থায় **Numbers** উপস্থাপন করে।

এখন প্রশ্ন— কেবল সংখ্যা নয়, আমরা যদি **Letters** (অক্ষর) উপস্থাপন করতে চাই—
তাহলে সেটা কীভাবে করবো?

তাহলে এখন প্রশ্ন— আমরা কীভাবে **letters** (বর্ণ/অক্ষর) উপস্থাপন করব?

ধরি, আমি একটা **text message** (টেক্সট মেসেজ) পাঠাচ্ছি।

এতে যদি লেখা থাকে “**A**” (capital letter A),

তবে কম্পিউটার সেটিকে কীভাবে বোঝে?

তবে আমরা একটি সমাধান নিতে পারি—

assign numbers to letters (অক্ষরগুলোর জন্য একটি করে সংখ্যা নির্ধারণ করি)।

এবং বাস্তবে ঠিক সেটাই করা হয়েছে।

একটি **standard system** তৈরি করা হয়েছে, যেটির নাম হলো **ASCII**
(**American Standard Code for Information Interchange**)।

এই সিস্টেমে, প্রতিটি **English character**-এর জন্য একটি নির্দিষ্ট **number** বরাদ্দ করা হয়েছে।

উদাহরণস্বরূপ:

- **A** → 65
 - **B** → 66
 - **C** → 67
- ...এভাবে ধারাবাহিকভাবে সব **capital letters**।

একইভাবে:

- **a** → 97
- **b** → 98

তবে কেউ যদি প্রশ্ন করেন, “**A** কেন 0 না হয়ে 65?”

উত্তর হলো— এইভাবে একসময় **standards committee** ঠিক করে দিয়েছে।

আমরা এখন কেবল সেই মান মেনে চলি।

এই **ASCII table**-এ কেবল **uppercase/lowercase letters**-ই নয়,
আরও আছে **digits (0–9)**, **punctuation marks** (., ?, :, ইত্যাদি),
এমনকি কিছু **control characters** (যেমন newline, tab)।

কিন্তু এখানে একটা সীমাবদ্ধতা ছিল।

ASCII মূলত **7-bit** বা **8-bit** এ কাজ করে,


এবং তাই এটি সর্বোচ্চ **128 থেকে 256 characters**-কে সমর্থন করে।

এটা **English language**-এর জন্য যথেষ্ট হলেও
বিশ্বের সব ভাষা, যেমন **Arabic, Bengali, Chinese, Japanese** ইত্যাদি—
এই সংখ্যায় প্রকাশ করা সম্ভব নয়।

তাই আজকাল আমরা ব্যবহার করি **Unicode**—
যেটি হলো **ASCII**-এর একটি **superset**।

Unicode অনেক বেশি **bits** ব্যবহার করে,
যেমন **16-bit, 24-bit**, বা **32-bit**।
এবং এভাবে এটি **4 billion characters** পর্যন্ত উপস্থাপন করতে পারে।

এমনকি এতে থাকে **emoji** (ইমোজি) পর্যন্ত!
কেননা ইমোজিও এখন **characters** হিসেবে বিবেচিত হয়।

যেমন, একটি জনপ্রিয় ইমোজি—
“” (face with tears of joy)
এর **Unicode number** হলো **4,036,991,106**।

এটি আসলে একটি **binary pattern**,
যেটিকে কম্পিউটার ইন্টারপ্রেট করে ছবির মতো দেখায়।

আর **different devices** (Android, iOS, Windows)
এই একই **emoji code** কে **different styles**-এ দেখায়—
কারণ প্রত্যেক কোম্পানির নিজস্ব **design team** থাকে।

তবে নিচের দিকের **older devices** বা **older fonts** না থাকলে
অনেক সময় আপনি ইমোজির জায়গায় **black box** বা **square** দেখেন।

এটি হয় কারণ আপনার **device** বা **font** আপডেট হয়নি।

এখন আমরা জানলাম—

Numbers, Letters, এমনকি **Emoji**-ও কম্পিউটার বোঝে **Zeros and Ones** (০ ও ১) এর মাধ্যমে।

পরবর্তী প্রশ্ন—
কম্পিউটার কীভাবে **Colors** (রং) উপস্থাপন করে?

তাহলে আমরা এখন জানি কীভাবে **numbers, letters**, এমনকি **emoji** পর্যন্ত **zeros and ones** দিয়ে উপস্থাপন করা যায়।

এবার প্রশ্ন— কীভাবে আমরা **color** (রং) উপস্থাপন করবো?

এটার জন্য আমরা ব্যবহার করি একটি পদ্ধতি যেটার নাম **RGB**।

RGB হলো তিনটি রঙের সংক্ষেপ:

- **Red** (লাল)
- **Green** (সবুজ)
- **Blue** (নীল)

প্রতিটি **pixel** (ডিসপ্লে এর একটি ক্ষুদ্রতম ডট)-এর রং বোঝাতে
কম্পিউটার **RGB values** ব্যবহার করে।

প্রতিটি **color component**-এর জন্য **1 byte** ব্যবহার হয়, অর্থাৎ ৮-বিট।

সুতরাং প্রতিটি **pixel** বোঝাতে ব্যবহার হয় **3 bytes** (বা ২৪-বিট):

- ১টি **Red** এর জন্য,
- ১টি **Green** এর জন্য,
- ১টি **Blue** এর জন্য।

প্রতিটি **component**-এর মান হতে পারে **0 থেকে 255** (যেহেতু ৮ বিট = $2^8 = ২৫৬$ মান সম্ভাব্য)।

উদাহরণস্বরূপ:

ধরি আপনি একটি রংকে এইভাবে সংজ্ঞায়িত করলেন—

- **Red = 72**
- **Green = 73**
- **Blue = 33**

এই তিনটি **byte** একসাথে একটা **color** তৈরি করে।

এখন আপনি যদি জিজ্ঞেস করেন, এটা কেমন রং?

→ এর মানে এই রং-এ থাকবে একটা **medium amount of red**,

একটা **medium amount of green**,

এবং একটু কম **blue**।

ফলে এটি হবে কিছুটা **yellowish** (হলুদের মতো) রঙ।

এখন আপনি যখন **Photoshop, Paint**, বা অন্য কোন **graphics software** ব্যবহার করেন,

তখন প্রতিটি **pixel** এভাবে একটি নির্দিষ্ট **RGB value** ধারণ করে।

আর এই **millions of pixels** একসাথে হয়ে তৈরি করে একটা **image**।

তাই আপনি যখন **digital photo** তোলেন— সেটি কেন এত **megabytes** হয়?

কারণ, প্রতি **pixel**-এর জন্য **3 bytes** দরকার হয়।

এবং একটি ছবিতে লাখ লাখ **pixels** থাকতে পারে।

তাই **image files** এত **large** হয়।

(বি.দ্র.: যদিও অনেক সময় **compression** ব্যবহার করে ছবিকে ছোট করা হয়।)

এখন আমরা **Color**-কে **bits** দিয়ে উপস্থাপন করতে শিখেছি।

তাহলে প্রশ্ন—

কীভাবে **Video** উপস্থাপন করা হয়?

আমরা এখন জানি কীভাবে **colors** (রং) **bits** দিয়ে উপস্থাপন করা যায়।

তাহলে, কীভাবে **videos** (ভিডিও) উপস্থাপন করা হয়?

ভেবে দেখুন, একসময় **videos**-কে বলা হতো **motion pictures** (চলমান ছবি)।

এবং সত্যিই, **videos** মানে হলো অনেকগুলো **images** বা **frames** যেগুলো একটার পর একটা দ্রুত দেখানো হয়।

এটা অনেকটা **flipbook** (পাতা উল্টানো বই)-এর মতো—
যেখানে একেক পৃষ্ঠায় একেক ছবি থাকে, আর দ্রুত পাতাগুলো উল্টালে মনে হয় চরিত্রটি নড়ছে।

Video-তেও ঠিক সেটাই হয়।

প্রতি **second** এ প্রায় **24 to 30 frames** (চিত্র) দেখানো হয়।

যেহেতু আমরা ইতোমধ্যে জেনেছি কীভাবে **images** উপস্থাপন করা হয়
(**pixels**, **RGB values**, ইত্যাদি),
তাই **videos** হলো কেবল এই **images**-এর একটি ধারাবাহিকতা।

এভাবে, প্রতিটি **frame** তৈরি হয় অসংখ্য **pixels** দিয়ে,
প্রতিটি **pixel**-এ আবার **3 bytes** (**RGB**) থাকে।
এবং প্রতি সেকেন্ডে যদি ৩০টি ফ্রেম হয়,
তাহলে ভিডিওর আকার বিশাল হয়।

এজন্যই **video files** এত **large**— অনেক সময় **gigabytes** (জীবিগা) হয়।

তাহলে আমরা কী কী শিখেছি?

- কীভাবে **numbers** উপস্থাপন করা হয়
- কীভাবে **letters** এবং **emoji** উপস্থাপন করা হয়
- কীভাবে **colors** এবং **images** তৈরি হয়
- এবং এখন, কীভাবে **video** উপস্থাপন করা হয়

এখন প্রশ্ন—

কীভাবে **sound** (ধ্বনি/সঙ্গীত) উপস্থাপন করা যায়?

তাহলে এখন প্রশ্ন—

কম্পিউটার কীভাবে **sound** (ধ্বনি/সংগীত) উপস্থাপন করে?

আপনি যদি একজন **musician** (সঙ্গীতশিল্পী) হন,
তাহলে আপনি জানেন— প্রতিটি **musical note** (সুর বা স্বর) এর রয়েছে নির্দিষ্ট **frequency** (কম্পন হার)।

কম্পিউটার কী করতে পারে?

→ প্রতিটি **number**-কে একটি **frequency** হিসেবে বোঝাতে পারে।

এভাবে, একেকটি **note** বা সুরকে একেকটি **digital value** দিয়ে বোঝানো যায়।

তবে শুধু **pitch** (স্বর) বা **frequency** যথেষ্ট নয়।

আমরা আরও কিছু উপাদান যোগ করতে পারি:

1. **Pitch** → কোন স্বর (frequency)
2. **Volume** → কত জোরে বাজছে
3. **Duration** → কতক্ষণ ধরে বাজছে

প্রতিটি **musical note** বোঝাতে আমরা ব্যবহার করতে পারি ৩টি **values**—

এবং প্রতিটি **value** সংরক্ষণে লাগবে **1 byte** বা তার বেশি।

ফলে, কম্পিউটার কী করে?

→ প্রতিটি **note**-এর জন্য সংরক্ষণ করে:

- **Pitch**
- **Volume**
- **Duration**
- এবং কখনো কখনো **Instrument type** (যেমন Piano, Violin)
যেটা বোঝাতে আরও একটি **value** সংরক্ষণ করা হয়।

এভাবে, **sound**-ও হয়ে যায় কেবল **patterns of zeros and ones**।

আপনি যখন **MP3, WAV**, বা অন্য কোন **audio file** প্লে করেন,
কম্পিউটার আসলে পড়ছে সেই **bits**,
যা নির্দেশ করে কোন সুর বাজবে, কত জোরে বাজবে, কতক্ষণ বাজবে—
সব কিছু।

সুতরাং এখন পর্যন্ত আমরা যা শিখেছি:

Information Type	Representation
Numbers	Binary (bits)
Letters	ASCII / Unicode
Images	RGB values (pixels)
Videos	Sequence of images (frames)
Sound	Pitch + Volume + Duration

Bottom line: সব কিছুর পেছনে রয়েছে শুধু **0s** এবং **1s**—
যেগুলোকে বলা হয় **bits**।

এবার প্রশ্ন হলো—

আমরা কীভাবে এই **bits** দিয়ে **problems solve** করবো?

এখন আমরা আলোচনা করব:

What is an Algorithm?

এখন আমরা অনেক কিছু শিখেছি—

Information কীভাবে **0s** এবং **1s** (bits) দিয়ে উপস্থাপন করা যায়:
সংখ্যা, অক্ষর, ছবি, ভিডিও, শব্দ— সব কিছু।

কিন্তু মূল বিষয় হচ্ছে—

এই **input** (ইনপুট) থেকে কীভাবে **output** (আউটপুট) তৈরি করা যায়?

এর মধ্যবর্তী ধাপে থাকে **Algorithm** (অ্যালগরিদম)।

তাহলে, What is an Algorithm?

একটি **Algorithm** হলো—

Step-by-step instructions (ধাপে ধাপে নির্দেশনা), যা দিয়ে আপনি একটি সমস্যা সমাধান করেন।

এটা হতে হবে খুবই **precise** (সুনির্দিষ্ট) এবং **unambiguous** (অস্পষ্টতাবিহীন)।

যেমন, ২৫ বছর আগে যখন আমি CS50 করতাম, তখন আমাদের শেখানো হয়েছিল

algorithm কীভাবে কাজ করে, সেটা বোঝাতে একজন প্রফেসর ক্লাসে দাঁড়িয়ে তার **beard trim** (দাড়ি ছাঁটা) করতেন।

কিন্তু আমি আজকে আপনাদের সামনে এনেছি অন্য একটি জিনিস—

একটি **phone book** (টেলিফোন ডাইরেক্টরি)।

এর মধ্যে থাকে হাজার হাজার মানুষের নাম এবং ফোন নম্বর, **alphabetical order**-এ সাজানো।

ধরি, আমি এই ফোনবুক থেকে "John Harvard"-এর নাম খুঁজছি।

প্রথম Algorithm:

আমি প্রথম পৃষ্ঠা থেকে শুরু করে একেকটা করে পৃষ্ঠা উল্টাচ্ছি।

- **Step 1:** প্রথম পৃষ্ঠা দেখো
- **Step 2:** নাম মিলছে কি না দেখো
- **Step 3:** না মিললে পরের পৃষ্ঠায় যাও

এটা ঠিক আছে, এবং এটা **correct algorithm**।

কিন্তু এটা খুবই **inefficient** (অদক্ষ)।

যদি নাম থাকে **Z** দিয়ে, তাহলে হাজার পৃষ্ঠা উল্টাতে হবে।

দ্বিতীয় Algorithm:

আমি **2 পৃষ্ঠা করে স্কিপ** করছি—

মানে, 2, 4, 6, 8, 10...

এটা কিছুটা দ্রুত, কিন্তু তবুও **flawed** (ত্রুটিপূর্ণ)।

কারণ আমি **miss** করে যেতে পারি "John Harvard" যদি সে দুটি পৃষ্ঠার মাঝখানে থাকে।

তবে যদি আমি যখন "J" ছাড়িয়ে যাই, তখন একটি পৃষ্ঠা পিছিয়ে যাচ্ছি— তাহলে তা **safer**।

তৃতীয় Algorithm:

এবার আসল **efficient algorithm**— যেটাকে আমরা বলি **Binary Search**।

আমি সরাসরি **book**-এর মাঝখানে যাচ্ছি।

→ যদি "M" পাই, বুঝি আমি বেশি এগিয়েছি, তাহলে বাম অর্ধেক রাখি।

→ যদি "F" পাই, বুঝি আমি পেছনে আছি, তাহলে ডান অর্ধেক রাখি।

→ এরপর বারবার অর্ধেক করে ভাগ করে যাই, যতক্ষণ না আমি **exact match** পাই বা বুঝতে পারি নামটি নেই।

Efficiency Comparison:

Algorithm	Steps (for n pages)
লাইন ধরে পড়া	n
দুই পৃষ্ঠা করে	$n / 2$
Binary Search	$\log_2(n)$

উদাহরণ:

যদি ফোনবুকে থাকে **1000** পৃষ্ঠা, তাহলে:

- লাইন ধরে \rightarrow 1000 ধাপ
- দুই করে \rightarrow 500 ধাপ
- Binary Search \rightarrow প্রায় **10** ধাপ! (কারণ $2^{10} = 1024$)

এই হলো **Algorithm-এর শক্তি**।

\rightarrow ভালো algorithm মানে কম সময়, কম effort, বেশি দক্ষতা।

এবং **Computer Science** এর একটি বড় লক্ষ্য হলো

“solving problems efficiently” (দক্ষভাবে সমস্যা সমাধান করা)।

তাহলে আমরা দেখলাম কীভাবে **binary search algorithm** (দ্বি-বিভাজন অনুসন্ধান পদ্ধতি) একটি সমস্যার সমাধানে **dramatic efficiency** (অসাধারণ দক্ষতা) এনে দেয়।

এখন প্রশ্ন—

আমরা কীভাবে এই **algorithm-গুলো code** (প্রোগ্রামিং কোড) হিসেবে লিখবো?

এর জন্য আমরা ব্যবহার করব একটি পদ্ধতি যেটিকে বলা হয় **pseudocode**।

What is Pseudocode?

Pseudocode হলো **English-like syntax** (ইংরেজির মতো দেখতে),

যেটা আসলে কোনো প্রোগ্রামিং ভাষা নয়,

কিন্তু বোঝায় কীভাবে একটি **algorithm** কাজ করে।

এটি **precise, clear**, এবং **logical**।

চলুন আমরা আমাদের **phone book search** উদাহরণটি **pseudocode** দিয়ে লিখি:

1. Pick up phone book
2. Open to middle of phone book
3. Look at page
4. If person is on page:
 5. Call person
6. Else if person is earlier in book:
 7. Open to middle of left half of book
 8. Go back to line 3
9. Else if person is later in book:

10. Open to middle of right half of book
 11. Go back to line 3
 12. Else:
 13. Quit
-

চলুন এই কোডটির গঠন দেখি:

- **"Call", "Open", "Look"** → এগুলো সব **functions** (কার্য)।
এগুলো এমন কাজ যা কম্পিউটার করবে।
 - **"If", "Else if", "Else"** → এগুলো হলো **conditionals** (শর্ত)।
এগুলো ব্যবহার করে আমরা বলি— কোন পরিস্থিতিতে কোন কাজ হবে।
 - **"Go back to line 3"** → এটা হলো **loop** (পুনরাবৃত্তি)।
মানে আবার আগের ধাপে ফিরে যাওয়া।
 - **"Person is earlier", "Person is on page"** → এগুলো হলো **Boolean expressions**—
যেগুলোর উত্তর হয় **True** বা **False**, অর্থাৎ **Yes** বা **No**।
-

এগুলো হচ্ছে **core building blocks**

যার ওপর দাঁড়িয়ে আছে সব **programming languages**:

- **Functions**
 - **Conditionals**
 - **Loops**
 - **Boolean logic**
-

এবার একটা গুরুত্বপূর্ণ প্রশ্ন—

"Go back to line 3" আবার আবার করলে কি **infinite loop** (অনন্ত লুপ) হতে পারে?

→ না। কারণ প্রতিবার আমরা **search space** অর্ধেক করে দিচ্ছি,
ফলে একসময় বই শেষ হয়ে যাবে এবং আমরা হয় **person খুঁজে পাবো**, নয় **quit** করবো।

এখন আমরা এই ধারণাগুলো ব্যবহার করেই ভবিষ্যতে বাস্তব কোড লিখবো—

প্রথমে **C**, পরে **Python, SQL, JavaScript** ইত্যাদি দিয়ে।

তবে তার আগে আসুন আমরা **real-world example** দেখি—

যেমন **Chatbot**, বা **AI (Artificial Intelligence)** কীভাবে কাজ করে।

এখন আমরা যেহেতু জানি কীভাবে **algorithm** তৈরি করি এবং সেটাকে **pseudocode** দিয়ে লিখি,

চলুন দেখি— কীভাবে একটি সাধারণ **Chatbot** বা **AI system** কাজ করে।

ধরি আপনি একটি **chatbot** বানাতে চান,
যেটা মানুষের সঙ্গে কথোপকথন করতে পারে।

আপনি তখন এইরকম **pseudocode** লিখতে পারেন:

```
If student says "hello":  
    Say "hello"  
Else if student says "goodbye":  
    Say "goodbye"  
Else if student says "how are you?":  
    Say "I am well"
```

এভাবে আপনি **conditionals** (শর্ত) দিয়ে নানান **input**-এর জন্য উত্তর লিখতে পারেন।

কিন্তু আপনি দুতই বুঝবেন—

মানুষের প্রশ্নের সংখ্যা **infinite** (অসীম)!

আপনি কি সত্যিই **every possible question**-এর জন্য আলাদা করে **"if"** লিখবেন?

→ সম্ভব নয়।

তাহলে AI (Artificial Intelligence) কীভাবে কাজ করে?

Traditional chatbot যেমন উপরের কোডে দেখেছি,

এগুলো **hard-coded rules**-এর ওপর নির্ভর করে।

কিন্তু **modern AI**, যেমন **ChatGPT** বা **CS50 Duck**,

এসব ব্যবহার করে **machine learning** এবং **neural networks**।

কীভাবে?

- আপনি **training data** দেন (যেমন Wikipedia, বই, ইন্টারনেটের লেখা)।
- এরপর **AI** শিখে নেয় কোন **input**-এর পরে কোন **output** আসতে পারে।
- সবকিছু হয় **probability-based** (সম্ভাবনার ভিত্তিতে)।

এগুলোকে বলে **Large Language Models (LLMs)**।

LLM-এর ভিতরে থাকে **neural networks**,

যেটা **biology** থেকে অনুপ্রাণিত—

আমাদের **brain**-এর **neurons** যেভাবে কাজ করে, সেটির অনুকরণে।

এই **neural network**-এ থাকে হাজার হাজার **nodes (neurons)**,

যারা একে অপরের সঙ্গে **connected** (সংযুক্ত) থাকে।

আপনি যখন একটা প্রশ্ন করেন,

তখন এটি সেই **connections** আর **probabilities**-এর মাধ্যমে সেরা উত্তর বের করে।

এবং এজন্যই, **modern AI** নির্দিষ্টভাবে কোড লেখা ছাড়াও অনেক কিছু "বুঝে ফেলতে পারে"।

কিন্তু এটি শুধুই সম্ভাবনার হিসাব—

কেউ একজন আলাদা করে সব উত্তর আগে থেকেই লিখে দেয়নি।

এবং সেই কারণে, আমরা **CS50**-তে একটি **AI assistant** ব্যবহার করি, যার নাম **The CS50 Duck** 🦆।

এর পিছনেও থাকে **AI**, তবে এটি ডিজাইন করা হয়েছে **CS50-এর প্রসঙ্গ বোঝার জন্য**,
এবং আপনাকে **guide** করার জন্য—not to give answers directly।
যেমনটি আপনি **Teaching Fellow** বা **TA**-এর কাছ থেকে আশা করবেন।

আপনারা ক্লাসে বা **Visual Studio Code (VS Code)**-এ এই **CS50 Duck** ব্যবহার করতে পারবেন।
তবে লক্ষ রাখবেন, সাধারণ **ChatGPT** বা অন্যান্য **AI tools** এই কোর্সে **allowed** নয়।
→ আপনি ব্যবহার করতে পারবেন <https://cs50.ai>
যেখানে **CS50 Duck** আপনাকে **debug, explain, help** করতে পারে।

পরবর্তী অংশে আমরা প্রথমবারের মতো আসল **code** দেখব—
যা লেখা হয়েছে **C programming language**-এ।
এটাই সেই প্রোগ্রাম যেখানে আমি নিজে **"minus 2"** পেয়েছিলাম:
একটি সাধারণ **"Hello, world" program**।?

আমরা এখন এমন একটি **code snippet** (কোডের টুকরো) দেখতে যাচ্ছি
যেটি লেখা হয়েছে **C programming language**-এ।

এটি দেখতে এমন:

```
#include <stdio.h>

int main(void)
{
    printf("hello, world\n");
}
```

এটি একটি খুবই সাধারণ প্রোগ্রাম যাকে বলা হয় **Hello, World Program**।
→ এর কাজ হলো কেবল **"hello, world"** টেক্সটটি প্রিন্ট করা।

তবে সত্যি বলতে, এটা দেখতে একটু **cryptic** (জটিল/অপরিচিত) মনে হতে পারে—
সব **punctuation**, যেমন **semicolon (;)**, **parentheses ()**, **curly braces {}**—
সবকিছু খুব নতুন লাগতে পারে।

তবে চিন্তা করবেন না—এসব কেবল **syntax** (বাক্য গঠন)।
আপনি যত বেশি **practice** করবেন, তত বেশি পরিচিত ও স্বচ্ছন্দ হয়ে উঠবেন।

সি (C) ভাষা কেন?

C হলো একটি **older** কিন্তু **very powerful** programming language।
→ এটা **fast**, এবং **low-level memory** নিয়ে কাজ করতে পারে।

আমরা এই কোর্সে **C** দিয়ে শুরু করবো কারণ:

- এটা আপনাকে **programming**-এর মূল ধারণা শেখাবে।
- এটি ভবিষ্যতে শেখার জন্য ভিত্তি তৈরি করবে: যেমন **Python, JavaScript**, ইত্যাদি।

C কঠিন হতে পারে শুরুতে, কিন্তু এর মাধ্যমে আপনি **how computers work under the hood** (কম্পিউটার ভিতরে ভিতরে কীভাবে কাজ করে) সেটা বুঝতে পারবেন।

কিন্তু আজকের জন্য...

আমরা আজ এইসব **syntax details** বাদ দিচ্ছি।

কারণ এসব **intellectually uninteresting** এখনই।

তার বদলে আজ আমরা ব্যবহার করব একটি **graphical programming language**—
যেটির নাম **Scratch** (MIT থেকে তৈরি)।

→ এটি **drag and drop blocks** দিয়ে প্রোগ্রাম লেখার পদ্ধতি।

Scratch হলো একটি **visual language**,
যেখানে আপনি কেবল **puzzle pieces** বা **blocks** টেনে এনে **code** তৈরি করতে পারেন।

Why Scratch?

- আপনাকে কোনো **semicolon** বা **brackets** চিন্তা করতে হবে না।
- আপনি **concepts** বোঝার ওপর ফোকাস করতে পারবেন।
- এটি নতুনদের জন্য দুর্দান্ত একটি **learning environment**।

এবং **problem set 0** (এই কোর্সের প্রথম অ্যাসাইনমেন্ট)-তে আপনাদের কাজ হবে **Scratch** ব্যবহার করে একটা প্রজেক্ট তৈরি করা।

চলুন দেখি কীভাবে **Scratch** কাজ করে।

আপনি যেতে পারেন scratch.mit.edu ওয়েবসাইটে।

সেখানে আপনি **Create** বাটনে ক্লিক করলে একটি **graphical programming interface** পাবেন,
যেখানে আপনি **blocks** (পাজল টুকরার মতো কোড) টেনে এনে **code** লিখতে পারবেন।

Scratch Interface:

1. Left Panel:

এখানে রয়েছে রঙ-বিভক্ত **categories of blocks**:

- **Motion** (নীল): চলাফেরা
- **Looks** (বেগুনি): দেখা বা লেখা
- **Sound** (গোলাপি): শব্দ
- ইত্যাদি

2. Middle Area:

এখানে আপনি **drag and drop** করে নিজের **script** তৈরি করবেন।

3. Right Panel:

এখানে থাকবে **sprite** (যেমন একটি বিড়াল),
যাকে আপনি নিয়ন্ত্রণ করবেন আপনার কোড দিয়ে।

উপরে থাকবে একটি **green flag** (চালু করার জন্য)

এবং একটি **red stop sign** (বন্ধ করার জন্য)।

Coordinates System (x, y):

Scratch Stage হলো একটি 2D screen, যেখানে প্রতিটি **sprite** থাকে একটি (x, y) অবস্থানে:

- মাঝখান = (0, 0)
- উপরের দিক = $y = +180$
- নিচের দিক = $y = -180$
- বামে = $x = -240$
- ডানে = $x = +240$

তবে আপনাকে সাধারণত এগুলো **manually** ব্যবহার করতে হয় না।

→ আপনি শুধু বলতে পারেন: “**move 10 steps**” বা “**turn 15 degrees**”।

প্রথম প্রোগ্রাম: "Say Hello, World"

Scratch-এ সবচেয়ে সাধারণ **starter program** হলো:

- **when green flag clicked**
- তারপর **say "hello, world"**

"Say" block হলো একটি **function**, যার **input** হলো “hello, world”।

এই **block** একধরনের **side effect** তৈরি করে—

→ স্ক্রীনে একটি **speech bubble** দেখা যায়।

এই প্রোগ্রামের **Input-Algorithm-Output** বিশ্লেষণ:

- **Input:** “hello, world”
 - **Algorithm:** say block
 - **Output:** স্ক্রীনে লেখা “hello, world”
-

এবার চলুন এই প্রোগ্রামটিকে আরও **interactive** করি।

আমরা চাই যে **user** যদি তার নাম টাইপ করে, তখন স্ক্রীনে লেখা আসবে:

→ **"Hello, David"** (বা যেই নামই টাইপ করুক)

এর জন্য আমরা ব্যবহার করবো:

- **ask [what's your name?] and wait** → input নেয়

- **answer block** → এটি হলো **return value**
- **join block** → দুটি লেখা **concatenate** (জুড়ে দেওয়া) করে

এভাবে আপনি তৈরি করতে পারেন:

"Hello, " + answer → যা output হিসেবে দেখানো হবে।

আপনি চাইলে **say block**-এর বদলে **text-to-speech extension** যোগ করে **speak block** ব্যবহার করতে পারেন—

যেটা শব্দে "Hello, David" বলে দেবে।

এইভাবে Scratch দিয়ে আপনি:

- **Variables** (যেমন: answer, score)
 - **Loops** (repeat, forever)
 - **Conditionals** (if, if-else)
 - **Events** (when clicked, when key pressed)
- সব কিছু ব্যবহার করতে পারবেন।

তাহলে আমরা ইতোমধ্যে একটা **interactive program** তৈরি করেছি, যেখানে **user** তার নাম লিখলে **sprite** বলে—

"Hello, David" (বা যে নামই হোক)।

এবার আমরা চাই **sprite** যেন **meow** (বিড়ালের ডাক) দেয়—

আর সেটি বারবার করতে পারে।

Repetition / Loop:

প্রথমে আপনি করতে পারেন:

- একই **"play sound meow" block** একাধিকবার **copy-paste** করে দেওয়া।
→ যেমন: তিনবার মেও দেওয়ার জন্য তিনটি **block** ব্যবহার করা।

কিন্তু এটি **poor design**— কারণ:

- আপনি যদি পরে সময় পরিবর্তন করতে চান,
তাহলে সবগুলো জায়গায় গিয়ে আলাদা আলাদা করে পরিবর্তন করতে হবে।

→ সমাধান: **loop** ব্যবহার করুন

Using Loop:

Scratch-এ **control** বিভাগে রয়েছে **repeat block**:

- আপনি বলতে পারেন:
→ **repeat 3 times: play sound meow and wait 1 second**

এভাবে **code** হয়:

```
when green flag clicked  
repeat 3  
  play sound "meow"  
  wait 1 seconds
```

→ এটা আগের তিনটি ব্লকের চেয়ে **better design**।

Custom Blocks / Functions:

ধরি, আপনি বারবার **meow** করাতে চান—

তাহলে কেন না একটি নিজের **custom block** তৈরি করা হয়?

→ যান **My Blocks** এ, ক্লিক করুন “**Make a Block**”

নাম দিন:

→ **meow n times**

এখন এই **block**-এ আপনি লিখবেন:

- **repeat n**
 - **play sound meow**
 - **wait 1 second**

এভাবে আপনি এখন কেবল লিখে দিতে পারেন:

```
when green flag clicked  
meow 3 times
```

→ এখন যদি আপনি ১০ বার মেও করাতে চান, শুধু লিখবেন **meow 10 times**

Inputs এবং Arguments:

এখানে **n** হলো একটি **input** বা **argument**,
যা আপনি **function** বা **block**-এ পাঠাচ্ছেন।

এটি **variables** এর মতোই—

তবে এটি **function**-এর **ভিতরে** ব্যবহৃত হয়।

Final Design Benefits:

- সহজে **reuse** করা যায়
 - কম **duplication**
 - পরবর্তীতে **maintenance** সহজ
 - কোড **clean** এবং **readable**
-

এইভাবে আপনি **abstraction** শিখছেন—
যেখানে আপনি একটি কাজ একবার **define** করেন,
তারপর শুধু ব্যবহার করেন— না জেনে কীভাবে সেটা কাজ করছে।

→ এটি **Computer Science**-এর একটি **core principle**।

এবার আমরা চাই আমাদের প্রোগ্রামটি আরও **interactive** হোক—
মানে, **user** যখন কিছু করে, তখন **sprite** যেন প্রতিক্রিয়া দেখায়।

Mouse Interaction:

Scratch-এ আপনি ব্যবহার করতে পারেন:

- **forever loop**
- তার ভিতরে একটি **if condition**
- এবং **sensing block** → “touching mouse pointer?”

উদাহরণ:

```
when green flag clicked
forever
  if <touching mouse pointer?>
    play sound "meow"
```

এখানে **sprite** (যেমন: বিড়াল) তখনই **meow** করবে,
যখন আপনি **mouse pointer** নিয়ে তার গায়ে টাচ করবেন।

→ এটি বাস্তবে “petting the cat”-এর মতো।

Video Sensing:

Scratch-এ আরেকটি **extension** আছে: **Video Sensing**

→ এটি আপনার **webcam** ব্যবহার করে দেখতে পারে আপনি স্ক্রিনে নড়াচড়া করছেন কি না।

এখানে আপনি ব্যবহার করতে পারেন:

- **when video motion > [number]**
- → তারপর একটি **sound** বা **animation trigger** করতে পারেন।

উদাহরণ:

```
when video motion > 10
  play sound "meow"
```

এভাবে আপনি বাস্তব জগতে হাত নাড়ালেই বিড়াল **meow** করতে পারে!

→ এটি **computer vision**-এর একটি সহজ সংস্করণ।

Events:

Scratch-এ এমন অনেক **event blocks** আছে:

- when green flag clicked
- when this sprite clicked
- when key pressed
- when backdrop switches to...
- when video motion > ...

এসব **event-driven programming**-এর মূল—
যেখানে প্রোগ্রাম বসে থাকে **"waiting for an event"**,
আর কোনো কিছু ঘটলে **respond** করে।

→ এই ধরনের **reactive behavior**-ই **modern user interface** এবং **games**-এর ভিত্তি।

Conclusion of This Section:

এখন পর্যন্ত আপনি শিখেছেন:

- কিভাবে **sprite** কথা বলে (say, speak)
- কিভাবে **loop** ও **custom block** দিয়ে কাজ সহজ হয়
- কিভাবে **interaction** বাড়ানো যায় **mouse** বা **video motion** দিয়ে
- এবং কীভাবে **event-driven thinking** কাজ করে

→ এসবের মাধ্যমে আপনি শুধু **coding** নয়,
thinking like a computer scientist শুরু করছেন।

এখন চলুন দেখে নিই কিছু **real projects**,
যেগুলো তৈরি করা হয়েছে **Scratch** ব্যবহার করে—
একটি হলো আমার নিজের প্রথম তৈরি করা প্রজেক্ট।

এই প্রজেক্টটির নাম ছিল **"Oscartime"**।

এটি একটি ছোট **game** ছিল,
যেখানে ব্যবহারকারীকে **trash** (আবর্জনা)
টেনে **Oscar the Grouch**-এর **trash can**-এ ফেলতে হতো।

🎵 [ব্যাকগ্রাউন্ডে গান: "I Love Trash"] 🎵

কীভাবে কাজ করে?

Sprites:

- Oscar (সাথে trash can)
- Random falling trash (jars, cans, shoes ইত্যাদি)
- প্রত্যেকটি **sprite** আলাদা **script** দ্বারা চালিত

Logic:

- প্রতি সেকেন্ডে একেকটি **trash** উপরের দিক থেকে নিচে নেমে আসে
- ব্যবহারকারী **mouse** দিয়ে সেটি ধরে **drag** করে
- যদি সেটা **Oscar's trash can**-কে স্পর্শ করে,
→ তখন সেটি উপরের একটি **random position** থেকে আবার পড়ে
→ এবং **score** ১ পয়েন্ট বাড়ে

Scratch Features ব্যবহৃত:

- **Motion blocks:** trash নিচে নেমে আসে
- **Sensing:** "touching Oscar?" → তখনই teleport হয় উপরে
- **Variables:** score গণনা রাখে
- **Costumes:** Oscar-এর মুখ খুলে-বন্দ হয়
- **Sound blocks:** Oscar এর থিম সং ও **meow, clank** ইত্যাদি

এটি তৈরি করতে সময় লেগেছিল,
কারণ আমি **step-by-step build** করেছিলাম:

1. প্রথমে শুধু একটি trash sprite নিচে পড়ছে
2. এরপর drag করলে উপরে ফিরে যাচ্ছে
3. এরপর score count যোগ
4. তারপর Oscar এর মুখ খুলছে যখন স্পর্শ হয়
5. এরপর গান, নতুন level, আরও sprites ইত্যাদি

→ এইভাবে **incremental development** করে আপনি জটিল project সহজে বানাতে পারেন।

এরপর আমরা দেখি আরেকটি প্রজেক্ট—

"Ivy's Hardest Game",

যেটি তৈরি করেছিল আমাদেরই একজন সাবেক ছাত্র।

এটি একটি **maze game** ছিল,

যেখানে আপনি একটি **Harvard logo** কে নিয়ন্ত্রণ করে যেতে হবে **exit** পর্যন্ত,

এবং পথের মধ্যে থাকবে চলমান **Yale** বা **MIT** এর বাধা।

Game Mechanics:

- **Arrow keys** দিয়ে চরিত্র চালানো
- **Collision detection:** দেয়ালের সঙ্গে বা Yale/MIT ছুঁলে শুরুতে ফিরে যাওয়া
- **Levels:** একেক লেভেলে নতুন **obstacle**
- **Custom blocks:** "feel for wall", "listen for keyboard" ইত্যাদি

এইসব প্রজেক্টের মাধ্যমে আপনি শিখবেন—

- **Problem Decomposition** (সমস্যা টুকরো করা)
- **State management** (পরিবর্তনশীল অবস্থা হ্যান্ডেল করা)
- **Events and interaction**
- এবং সবচেয়ে গুরুত্বপূর্ণ: **computational creativity**

আপনারা যারা এখন Scratch-এ নতুন—

চিন্তা করবেন না।

আমরা চাই, আপনি ধাপে ধাপে **build** করুন—

একবারে পুরা গেম বা অ্যাপ বানাতে হবে না।

→ মনে রাখবেন, **simple beginnings** দিয়েই **big ideas** তৈরি হয়।

নিচে CS50 Lecture 0-এর পরবর্তী অংশ বাংলায় অনুবাদ করা হলো, যেখানে ইংরেজি মূল শব্দগুলো ইংরেজিতেই রাখা হয়েছে এবং বাংলায় ব্যাখ্যা দেওয়া হয়েছে:

DAVID MALAN:

এখন আমরা আরেকটি গেমের দিকে তাকাই, যার নাম “**Ivy’s Hardest Game**”।

এটি একটি **maze-style game**,

যেখানে আপনাকে **Harvard logo sprite** ব্যবহার করে

এক কোণা থেকে আরেক কোণায় পৌঁছাতে হয়।

তবে চ্যালেঞ্জ হলো— মাঝপথে আছে **obstacles**,

যেমন **Yale** ও **MIT logos**, যেগুলো গেমের চরিত্রের দিকে এগিয়ে আসে।

Movement এবং Boundary:

প্রথমে, শুধুমাত্র **Harvard sprite** রয়েছে, যেটি আপনি **arrow keys** দিয়ে চালাতে পারেন।

এর জন্য কোড কিছুটা এমন:

```
forever
  if key "up arrow" pressed then change y by 1
  if key "down arrow" pressed then change y by -1
  if key "left arrow" pressed then change x by -1
  if key "right arrow" pressed then change x by 1
```

এছাড়া, যদি **sprite** কোনো দেয়ালের সাথে **touch** করে,

তবে সেই দিক থেকে সরিয়ে নেওয়া হয়।

এটি করা হয় একটি **custom block** দিয়ে, যেমন:

feel for walls

Yale Enemy:

এরপর যুক্ত হয় **Yale sprite**,
যেটি **left to right** চলাচল করে স্বয়ংক্রিয়ভাবে।

এই চলাচলের জন্য ব্যবহৃত হয়:

- **point in direction 90**
- **move steps**
- **if touching edge then turn 180 degrees**

এভাবে Yale বারবার **bounce** করে।

MIT Enemy:

এরপর যুক্ত হয় **MIT sprite**,
যেটি সরাসরি **Harvard sprite**-কে অনুসরণ করে।

এর জন্য কোড কিছুটা এরকম:

```
forever
  point towards [Harvard sprite]
  move 1 steps
```

→ এটি তৈরি করে **chasing effect**, যেখানে MIT প্রতিনিয়ত Harvard কে ধরতে চায়।

MIT-কে যদি আপনি **faster** করতে চান,
তবে **move 1 steps** এর বদলে **move 2 steps** বা **move 10 steps** করতে পারেন।

Game Structure:

গেমটি ধাপে ধাপে উন্নত করা হয়েছে:

1. প্রথমে কেবল একটিমাত্র গন্ডি (wall)
 2. এরপর যোগ হয় Yale
 3. তারপর আরও Yale, এবং MIT
 4. শেষে পুরো গেম হয় একটি **level-based chase game**,
যেখানে প্রতিবার **exit point**-এ পৌঁছালে নতুন **level** শুরু হয়
-

এইসব দিয়ে বোঝানো হয়:

- **Control structures** (loop, conditionals)
 - **Event handling** (when key pressed, when touching)
 - **Abstraction** (custom blocks দিয়ে)
 - এবং **incremental development**— যা একটি বড় ধারণা, ধাপে ধাপে তৈরি হয়
-

লেকচারের শেষে, আমরা সবাইকে আমন্ত্রণ জানাই আমাদের ঐতিহ্যবাহী **CS50 Cake Reception**-এ,
যেটি হয় **Transept**-এ।

→ এইভাবেই শুরু হয় CS50 যাত্রা: শিখে, বানিয়ে, মজা করে।

This was CS50. Welcome aboard. 🎉