

Lesson 1

Tokens and Lexing

Covered in this lesson:

- Lexer and Lexing (tokenizing)
- Token types: keywords and identifiers

What you write, is what the computer sees. So if you write the below code, the computers also sees it. But it cant understand its meaning.

```
LET X = 5  
PRINT X + 2
```

So we do this: Characters → Tokens → Structure → Meaning → Execution

The characters are what you've written.

In order to make tokens, we will undergo the process of *Lexing*.

LEXING (Tokenizing)

It groups characters into **tokens**.

What Is a Token?

A token has two things:

type - what kind?

text - what value?

Character/Text	Token
LET	KEYWORD
X	IDENTIFIER
=	EQUAL
5	NUMBER

The Idea Behind A Lexer

The lexer walks through the code one character at a time

LET X = 5

L → E → T → space → X → space → = → space → 5

At each step, it asks:

Is this a letter?

Is this a digit?

Is this whitespace?

Is this a symbol?

Then it decides what token to make.

Token: Identifier Vs Keywords

Look at the words below:

LET

X

PRINT

They all look the same. They are all composed of letters, right? So how do the computer know if it is an identifier or a keyword? Well, first we'll have to define our keywords?

Example of keywords: LET, END, PRINT, FOR

After defining the keywords, the computer will do this:

- Read the word
- Compare it to known keywords.
- If word is defined in keyword → keyword.
- Else → identifier.

Below are the important functions that a Lexer class has:

IDEA	MEANING
source	the whole file text
pos	where you are. known as index in other programming languages
peek()	look without moving.

	looks at the current character
advance()	move forward. this moves the pointer to the next position
skipwhitespace()	ignore spaces
nexttoken()	produce one token

Let's take a look at how the lexer class functions:

This is only a peek of what's in the next lesson. On where we start to build a lexer.

```
class Lexer {
    string source;
    int pos;
};
```

How does it know if it has reached the end of the source(code)?

```
bool Lexer::isAtEnd(){
    return pos >= source.size()
}
```

look at the current character:

```
current = source[pos]
```

look for a character next to the current position:

```
char peek(){
    return source[pos+1]
};
```

How does it move forward?

```
char advance(){
    pos++
};
```

How to ignore spaces?

```
while (current == ' ') advance();
```