Part 3

Lexing

The Input Problem

- At some point we need to read source code

```
/* Print the first ten factorials */
var n int = 1;
var value int = 1;

while n < 10 {
    value = value * n;
    print value ;
    n = n + 1;
}
```

- This is the "parsing" problem.
- Goal: source → model

Tokenizing and Parsing

- Parsing is often subdivided into two problems
  - Tokenizing (aka. "Lexing")
  - Parsing
- In this part: Lexing

Lexing in a Nutshell

- Convert input text into a token stream

```
b = 40 + 20*(2+3)/37.5

NAME = NUM + NUM * ( NUM + NUM ) / FLOAT
```

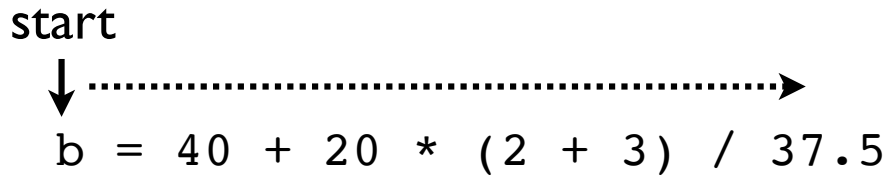
- A token is a typed text string.

```
b    ---->   ('NAME','b')

=    ---->   ('ASSIGN','=')

40   ---->   ('NUM','40')
```

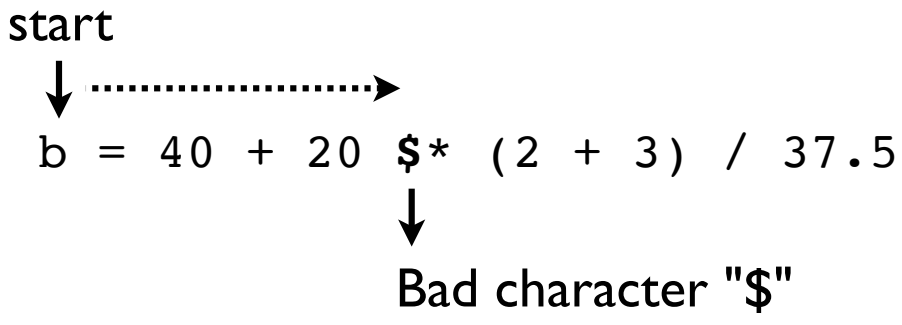
- Question: How to do it?

Text Scanning

- Perform a linear text scan

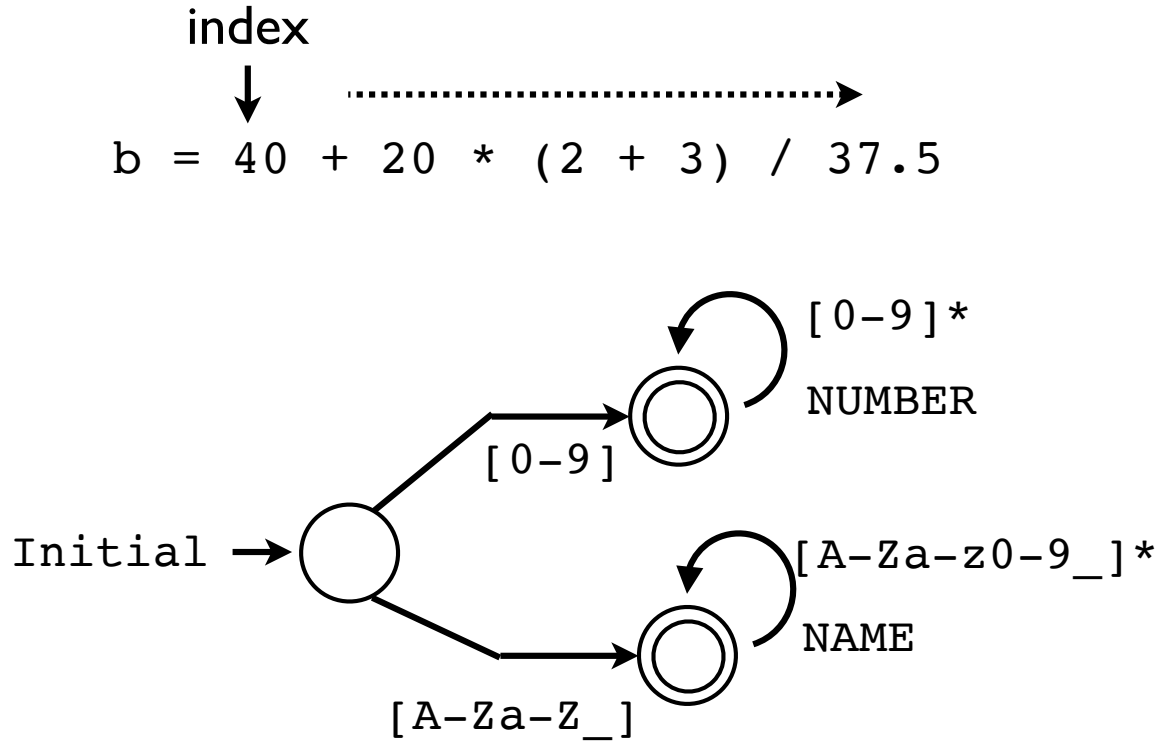


- ALL characters are consumed
- Otherwise error:



Text Recognition

- Scanning is based on matching patterns



- Could use regex, but also code by hand

Rough Coding Template

```
text = "... source code ..."
index = 0

while index < len(text):    # scan left-to-right
    if text[index].isdigit():
       # A number
       start = index
       while text[index].isdigit():
           index += 1
       yield ('NUMBER', text[start:index])

    elif text[index].isalpha() or text[index] == '_':
       # A name
       start = index
       while text[index].isalnum() or text[index] == '_':
           index += 1
       yield ('NAME', text[start:index])
    ...
```

Commentary

- Tokenizing is NOT an interesting problem in the context of modern compiler writing
- Yes, it is an essential part of parsing.
- But, it's hardly the most important thing.

Project

- Find the file `wabbit/tokenize.py`
- Follow instructions inside
- Will group code part of it