

# HW 3 Problem 2

Question: Write a simple parser for the following EBNF grammar:

```
<S> ::= { a } <X> | b <X>  
- <X> ::= c | d
```

Provided test cases:

1. Input: "bc"  
Print: "Input is valid"
2. Input: "acd"  
Print: "Syntax error at character position 2"
3. Input: "aaad"  
Print: "Input is valid"
4. Input: "c"  
Print: "Input is valid"
5. Input: "2yz"  
Print: "Syntax error at character position 0"
6. Input: "" (empty)  
Print: "Syntax error at character position 0"

Note: First language is Python

**Algorithm/Pseudo code:**

```
1  class SimpleParser:  
2      constructor (accepts input string):  
3          input = input string  
4          char_pos = 0  
5  
6      def fun_s():  
7          try block:  
8              check for empty string, raise exception if it is  
9              if letter is a, we need to call fun_x()  
10             else if letter is b, we need to increment char_pos and call fun_x()  
11             else we call fun_x  
12         catch all errors  
13  
14     def fun_x():  
15         if current char is a c or d:  
16             if we are at the end of the string:  
17                 input is valid  
18         else  
19             increment char_pos  
20             raise exception  
21     else:  
22         raise exception
```

## Actual Code:

```
1 class SimpleParser:
2     def __init__(self, s):
3         self.input = s
4         self.char_pos = 0
5
6     def fun_s(self):
7         try:
8             if len(self.input) == 0:
9                 raise Exception("Syntax error at character position " + self.char_pos)
10            letter = self.input[self.char_pos]
11            if letter == 'a':
12                self.fun_x()
13            elif letter == 'b':
14                self.char_pos += 1
15                self.fun_x()
16            else:
17                self.fun_x()
18        except Exception as error:
19            print(repr(error))
20
21    def fun_x(self):
22        if (self.input[self.char_pos] == 'c' or self.input[self.char_pos] == 'd'):
23            if self.char_pos == len(self.input) - 1:
24                print("Input is valid")
25            else:
26                # acd
27                self.char_pos += 1
28                raise Exception("Syntax error at character position " + self.char_pos)
29        else:
30            raise Exception("Syntax error at character position " + self.char_pos)
```

## Syntax Error:

- When I raise an exception, I need to convert self.char\_pos to a string when I create the error message

## Working Code

```
1 class SimpleParser:
2     def __init__(self, s):
3         self.input = s
4         self.char_pos = 0
5
6     def fun_s(self):
7         try:
8             if len(self.input) == 0:
9                 raise Exception("Syntax error at character position " + str(self.char_pos))
10            letter = self.input[self.char_pos]
11            if letter == 'a':
12                self.fun_x()
13            elif letter == 'b':
14                self.char_pos += 1
15                self.fun_x()
16            else:
17                self.fun_x()
18        except Exception as error:
19            print(repr(error))
20
21    def fun_x(self):
22        if (self.input[self.char_pos] == 'c' or self.input[self.char_pos] == 'd'):
23            if self.char_pos == len(self.input) - 1:
24                print("Input is valid")
25            else:
26                # acd
27                self.char_pos += 1
28                raise Exception("Syntax error at character position " + str(self.char_pos))
29        else:
30            raise Exception("Syntax error at character position " + str(self.char_pos))
```

## Debugging

1. Did not increment char\_pos before calling self.fun\_x() in line 12. To fix:

```
def fun_s(self):
    try:
        if len(self.input) == 0:
            raise Exception("Syntax error at character position " + str(self.char_pos))
        letter = self.input[self.char_pos]
        if letter == 'a':
            self.char_pos += 1
            self.fun_x()
        elif letter == 'b':
            self.char_pos += 1
            self.fun_x()
        else:
            self.fun_x()
    except Exception as error:
        print(repr(error))
```

2. Did not account for the fact that we can have multiple "A"s in a row. Added a while loop to account for multiple "A"s

```
-  
    if letter == 'a':  
        while letter == 'a' and self.char_pos < len(self.input):  
            letter = self.input[self.char_pos]  
            self.char_pos += 1  
        self.char_pos -= 1  
        # input string consists entirely of 'a's  
        if letter == 'a':  
            raise Exception("Syntax error at character position " + str(self.char_pos))  
        else:  
            self.fun_x()
```

3. Need to catch error involving multiple repeating "b"s

```
-  
    elif letter == 'b':  
        # only 1 b  
        if self.char_pos == len(self.input) - 1:  
            raise Exception("Syntax error at character position " + str(self.char_pos))  
        else:  
            self.char_pos += 1  
            self.fun_x()
```

## Add Documentation

```
1 class SimpleParser:
2     def __init__(self, s):
3         self.input = s
4         self.char_pos = 0
5
6     def fun_s(self):
7         try:
8             # if input is empty string
9             if len(self.input) == 0:
10                 raise Exception("Syntax error at character position " + str(self.char_pos))
11             letter = self.input[self.char_pos]
12             if letter == 'a':
13                 # loop through repeated As
14                 while letter == 'a' and self.char_pos < len(self.input):
15                     letter = self.input[self.char_pos]
16                     self.char_pos += 1
17                 self.char_pos -= 1
18                 # input string consists entirely of 'a's
19                 if letter == 'a':
20                     raise Exception("Syntax error at character position " + str(self.char_pos))
21                 else:
22                     self.fun_x()
23             elif letter == 'b':
24                 # only 1 b
25                 if self.char_pos == len(self.input) - 1:
26                     raise Exception("Syntax error at character position " + str(self.char_pos))
27                 else:
28                     self.char_pos += 1
29                     self.fun_x()
30             else:
31                 self.fun_x()
32         except Exception as error:
33             print(repr(error))
34
35     def fun_x(self):
36         if (self.input[self.char_pos] == 'c' or self.input[self.char_pos] == 'd'):
37             # If exception not prev thrown and input ends with c or d, it is valid
38             if self.char_pos == len(self.input) - 1:
39                 print("Input is valid")
40             else:
41                 # If there are more letters after c/d (acd)
42                 self.char_pos += 1
43                 raise Exception("Syntax error at character position " + str(self.char_pos))
44         # any char other than c or d
45         else:
46             raise Exception("Syntax error at character position " + str(self.char_pos))
47
```

**Extra Test Cases Used for Debugging:**

1. Input: "abd"  
Print: "Syntax error at character position 1"
2. Input: "b"  
Print: "Syntax error at character position 0"
3. Input: "aaaa"  
Print: "Syntax error at character position 3"
4. Input "bbbb"  
Print: "Syntax error at character position 1"
5. Input: "cc"  
Print: "Syntax error at character position 1"
6. Input: "3"  
Print: "Syntax error at character position 0"

Note: Second language is Rust

### Actual Code:

```
1  extern crate custom_error;
2  use custom_error::custom_error;
3
4  custom_error!{InvalidSyntaxError
5  |   Bad{pos:i32} = "Syntax error at character position {pos}"
6  }
7
8  struct SimpleParser {
9      input: String,
10     char_pos: i32,
11     input_len: i32
12 }
13
14 impl SimpleParser {
15     fn new(user_input: &str) -> SimpleParser {
16         SimpleParser {
17             input: user_input.to_string(),
18             char_pos: 0,
19             input_len: user_input.to_string().chars().count() as i32
20         }
21     }
22 }
```

```
fn fun_s(&mut self) {
    if self.input_len == 0 {
        println!("Syntax error at character position {}", self.char_pos);
        return;
    }
    let mut letter:char = self.input.chars().nth(self.char_pos).unwrap();
    if letter == 'a'{
        while letter == 'a' && self.char_pos < self.input_len {
            letter = self.input.chars().nth(self.char_pos).unwrap();
            self.char_pos += 1;
        }
        self.char_pos -= 1;
        if letter == 'a' {
            println!("Syntax error at character position {}", self.char_pos);
            return;
        }
    } else if letter == 'b' {
        if self.char_pos == self.input_len - 1 {
            println!("Syntax error at character position {}", self.char_pos);
            return;
        } else {
            self.char_pos += 1;
        }
    }
}
```

```

51     fn fun_x(&mut self) -> Result<(), InvalidSyntaxError> {
52         let letter:char = self.input.chars().nth(self.char_pos).unwrap();
53         if letter == 'c' || letter == 'd' {
54             if self.char_pos == self.input_len - 1 {
55                 return Ok(());
56             } else {
57                 self.char_pos += 1;
58                 return Err(InvalidSyntaxError::Bad{pos: self.char_pos});
59             }
60         } else {
61             return Err(InvalidSyntaxError::Bad{pos: self.char_pos});
62         }
63     }
64 }
65

```

## Syntax Error:

1. Whenever I am indexing the input string, I need to convert self.char\_pos to usize using self.char\_pos as usize

## Working Code

```

1  custom_error!{InvalidSyntaxError
2  |   Bad{pos:i32} = "Syntax error at character position {pos}"
3  | }
4
5
6
7  struct SimpleParser {
8      input: String,
9      char_pos: i32,
10     input_len: i32
11 }
12
13 impl SimpleParser {
14     fn new(user_input: &str) -> SimpleParser {
15         SimpleParser {
16             input: user_input.to_string(),
17             char_pos: 0,
18             input_len: user_input.to_string().chars().count() as i32
19         }
20     }
21

```



```

22 fn fun_s(&mut self) {
23
24     if self.input_len == 0 {
25         println!("Syntax error at character position {}", self.char_pos);
26         return;
27     }
28     let mut letter:char = self.input.chars().nth(self.char_pos as usize).unwrap();
29     if letter == 'a'{
30         while letter == 'a' && self.char_pos < self.input_len {
31             letter = self.input.chars().nth(self.char_pos as usize).unwrap();
32             self.char_pos += 1;
33         }
34         self.char_pos -= 1;
35         if letter == 'a' {
36             println!("Syntax error at character position {}", self.char_pos);
37             return;
38         }
39     } else if letter == 'b' {
40         if self.char_pos == self.input_len - 1 {
41             println!("Syntax error at character position {}", self.char_pos);
42             return;
43         } else {
44             self.char_pos += 1;
45         }
46     }
47 }

```

```

49 fn fun_x(&mut self) -> Result<(), InvalidSyntaxError> {
50     let letter:char = self.input.chars().nth(self.char_pos as usize).unwrap();
51     if letter == 'c' || letter == 'd' {
52         if self.char_pos == self.input_len - 1 {
53             return Ok(());
54         } else {
55             self.char_pos += 1;
56             return Err(InvalidSyntaxError::Bad{pos: self.char_pos});
57         }
58     } else {
59         return Err(InvalidSyntaxError::Bad{pos: self.char_pos});
60     }
61 }
62 }

```

## Debugging

1. Threw errors using custom error, but I never caught the error. To fix, I need to add a match statement at the end of fun\_s()

```
match self.fun_x() {  
    Ok(_) => println!("Input is valid"),  
    Err(e) => println!("{}", e)  
}
```

## Add Documentation

```
1  extern crate custom_error;  
2  use custom_error::custom_error;  
3  
4  custom_error!{InvalidSyntaxError  
5  |   Bad{pos:i32} = "Syntax error at character position {pos}"  
6  | }  
7  
8  struct SimpleParser {  
9  |   input: String,  
10 |   char_pos: i32,  
11 |   input_len: i32  
12 | }  
13  
14 impl SimpleParser {  
15 |   fn new(user_input: &str) -> SimpleParser {  
16 |       SimpleParser {  
17 |           // initialize attributes  
18 |           input: user_input.to_string(),  
19 |           char_pos: 0,  
20 |           input_len: user_input.to_string().chars().count() as i32  
21 |       }  
22 |   }  
23 }
```

```

24 fn fun_s(&mut self) {
25     // Accounting for empty string
26     if self.input_len == 0 {
27         println!("Syntax error at character position {}", self.char_pos);
28         return;
29     }
30     // Get letter at current char_pos
31     let mut letter:char = self.input.chars().nth(self.char_pos as usize).unwrap();
32     if letter == 'a'{
33         // loop through repeating As if any
34         while letter == 'a' && self.char_pos < self.input_len {
35             letter = self.input.chars().nth(self.char_pos as usize).unwrap();
36             self.char_pos += 1;
37         }
38         self.char_pos -= 1;
39         // error if input string contains entirely of As
40         if letter == 'a' {
41             println!("Syntax error at character position {}", self.char_pos);
42             return;
43         }
44     } else if letter == 'b' {
45         // error if only one b
46         if self.char_pos == self.input_len - 1 {
47             println!("Syntax error at character position {}", self.char_pos);
48             return;
49         } else {
50             self.char_pos += 1;
51         }
52     }
53
54     // Catch errors that are thrown
55     match self.fun_x() {
56         Ok(_) => println!("Input is valid"),
57         Err(e) => println!("{}", e)
58     }
59 }

```

```
61     fn fun_x(&mut self) -> Result<(), InvalidSyntaxError> {
62         let letter:char = self.input.chars().nth(self.char_pos as usize).unwrap();
63         if letter == 'c' || letter == 'd' {
64             // If c/d is the last letter in input, it is valid
65             if self.char_pos == self.input_len - 1 {
66                 return Ok(());
67             } else {
68                 self.char_pos += 1;
69                 return Err(InvalidSyntaxError::Bad{pos: self.char_pos});
70             }
71             // Any char other than c or d
72         } else {
73             return Err(InvalidSyntaxError::Bad{pos: self.char_pos});
74         }
75     }
76 }
```