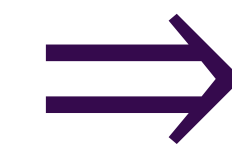
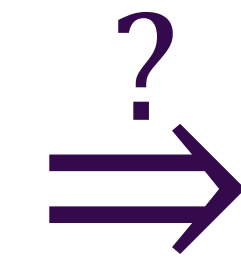


# What makes program synthesizers hard to learn for novice programmers?

**Domain experts**  
(Law, medicine, science, ...)



**No time to learn to code!**



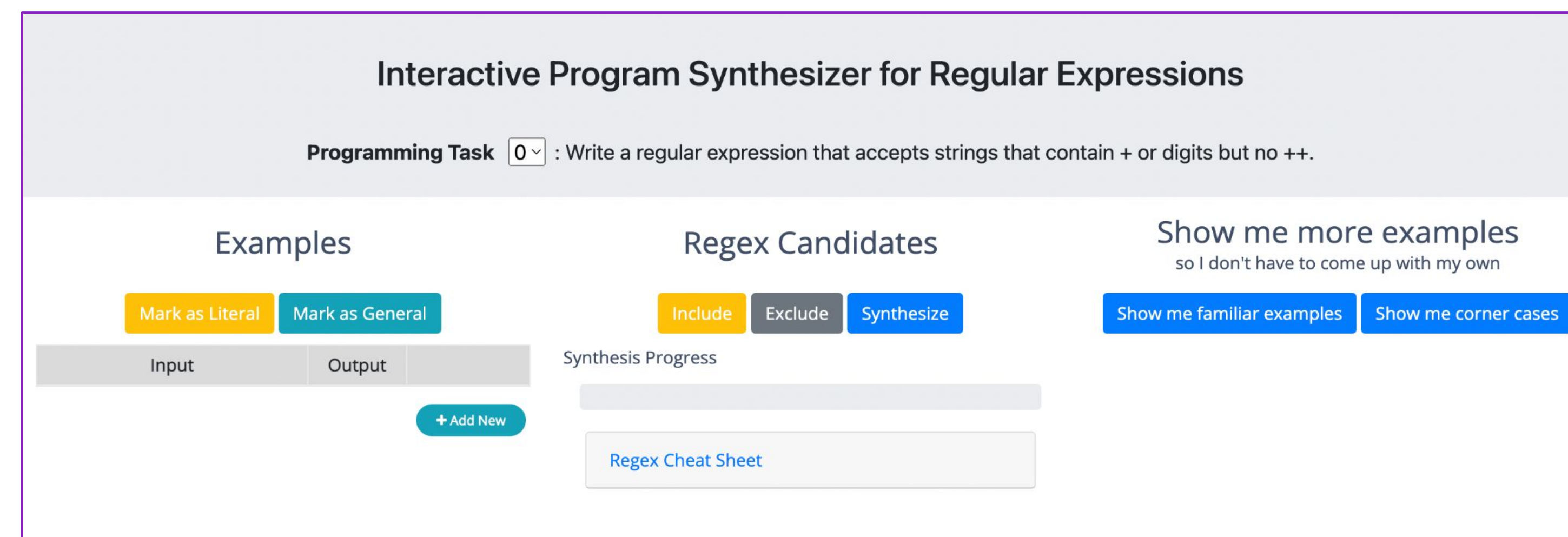
**Program synthesis**

Automatic generation of code that satisfies a user-provided **specification**

```

1 """
2 Abbreviate
3 Return the abbreviation of
4 the given name:
5 >>> task('Alan Turing') == 'A.T'
6 """
7 def task(name):
8     # 1. Split the name into words
9     # 2. Get the first letter of each
10    # 3. Put dots between them
11    abbr = ''
12    return abbr
13
14 task('Augusta Ada King')
15

```



```

public class Main2 {
    public static int latitude;
    public static int Y;

    Run | Debug
    public static void main(String[] args) {
        X = 0;
        // ...
    }

    public int[] getEnd() {
        int[] pos = new int[2];
        pos[0] = X + 5;
        pos[1] = Y + 5;
        return pos;
    }
}

```

	A	B	C
1	<b>Name</b>	<b>Last Name, First Name</b>	<b>Handle</b>
2	Calvin Canaday	Canaday, Calvin	ccanaday
3	Perla Lindstrom		
4	Velvet Blansett		
5	Danette Giles		
6	Maxwell Herren		
7	Barry Lombardi		

```

def count_letters(x):
    count = 0
    for i in x:
        if i.isalpha():
            count += 1
    return count

```

## Tool characteristics

## User behavior

Voluntary specification ↔ Incidental specification

User-triggered initiation ↔ Triggerless initiation

User-triggered result communication ↔ Triggerless result communication

Incorrectly believing the synthesizer made progress

Incorrectly believing the synthesizer did *not* make progress

What makes a good specification?

Understanding *learnability implications* of *tool characteristics* can help designers make *empirically-supported design decisions*

Understanding *user misconceptions* can help designers make systems to *proactively combat them*