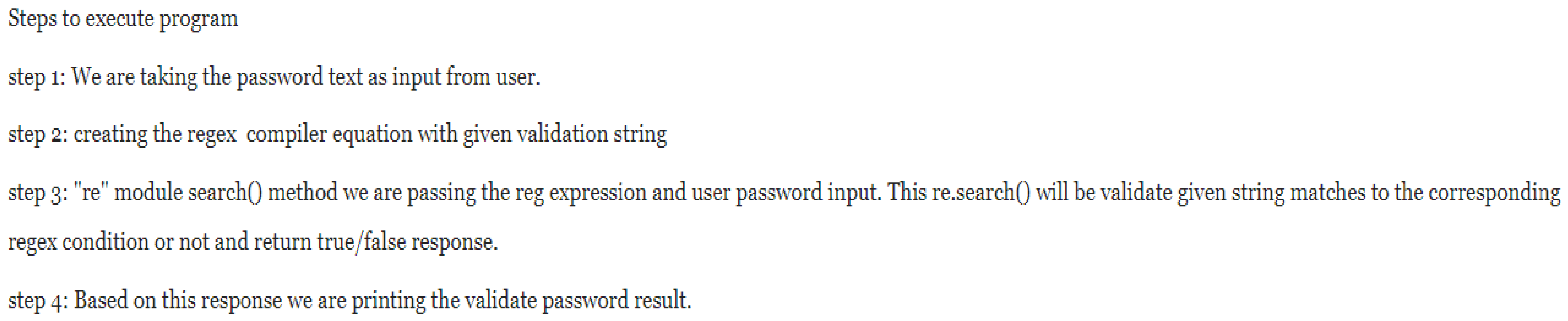
Regular Expression

1. re.purge()
2. Password and Email Validation (ReGex)
3. Metacharacters in Details

This example we are validating the password with regex equation which will validate below conditions

One Capital Letter
One Number
Special Character
Length Should be 8-18
Let's write python regex example to validate password

```
import re

# input
print("Please enter password should be \n1) One Capital Letter\n2) Special Character\n3) One Number \n4) Length Should be 8-18: ")
pswd = input()
reg = "^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*#?&])[A-Za-z\d@$!#%*?&]{8,18}$“

# compiling regex
match_re = re.compile(reg)
# searching regex
res = re.search(match_re, pswd)
# validating conditions
if res:
  print("Valid Password")
else:
  print("Invalid Password")
```



Steps to execute program

step 1: We are taking the password text as input from user.

step 2: creating the regex compiler equation with given validation string

step 3: "re" module search() method we are passing the reg expression and user password input. This re.search() will be validate given string matches to the corresponding regex condition or not and return true/false response.

step 4: Based on this response we are printing the validate password result.

Email Validation (Regular Expression)

```
import re

regex = re.compile(r'([A-Za-z0-9]+[.-_])*[A-Za-z0-9]+@[A-Za-z0-9-]+(\.[A-Z|a-z]{2,})+')

def isValid(email):
    if re.fullmatch(regex, email):
      print("Valid email")
    else:
      print("Invalid email")
```

Now, let's test the code on some of the examples we took a look at earlier:

```
isValid("name.surname@gmail.com")
isValid("anonymous123@yahoo.co.uk")
isValid("anonymous123@...uk")
isValid("...@domain.us")
```

This results in:

```
Valid email
Valid email
Invalid email
Invalid email
```

RE.PURGE() FUNCTION

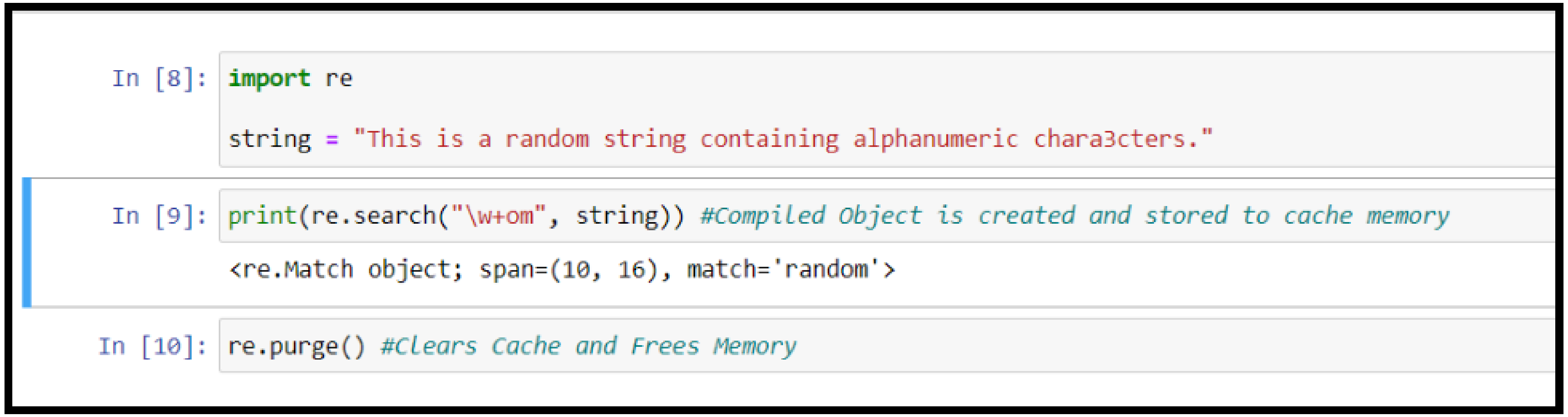
The **re.purge()** clears the regular expression cache. As you are aware, cache memory is a storage location which holds temporary data for applications and programs, so that it can be referenced later.

The regular expression cache stores compiled regular expression objects. These objects are created and stored when we use search functions like **re.match(), re.search()** directly by passing pattern into it.

If the search function is called multiple times, instead of creating/recompiling the object multiple times the engine compiles the object once and stores it in cache.

A more efficient approach would be to create a regex object in your program using **re.compile()** and storing it in a variable. Then using that object do search functions. This would not create unnecessary objects in the cache and eliminate the use of **re.purge().**

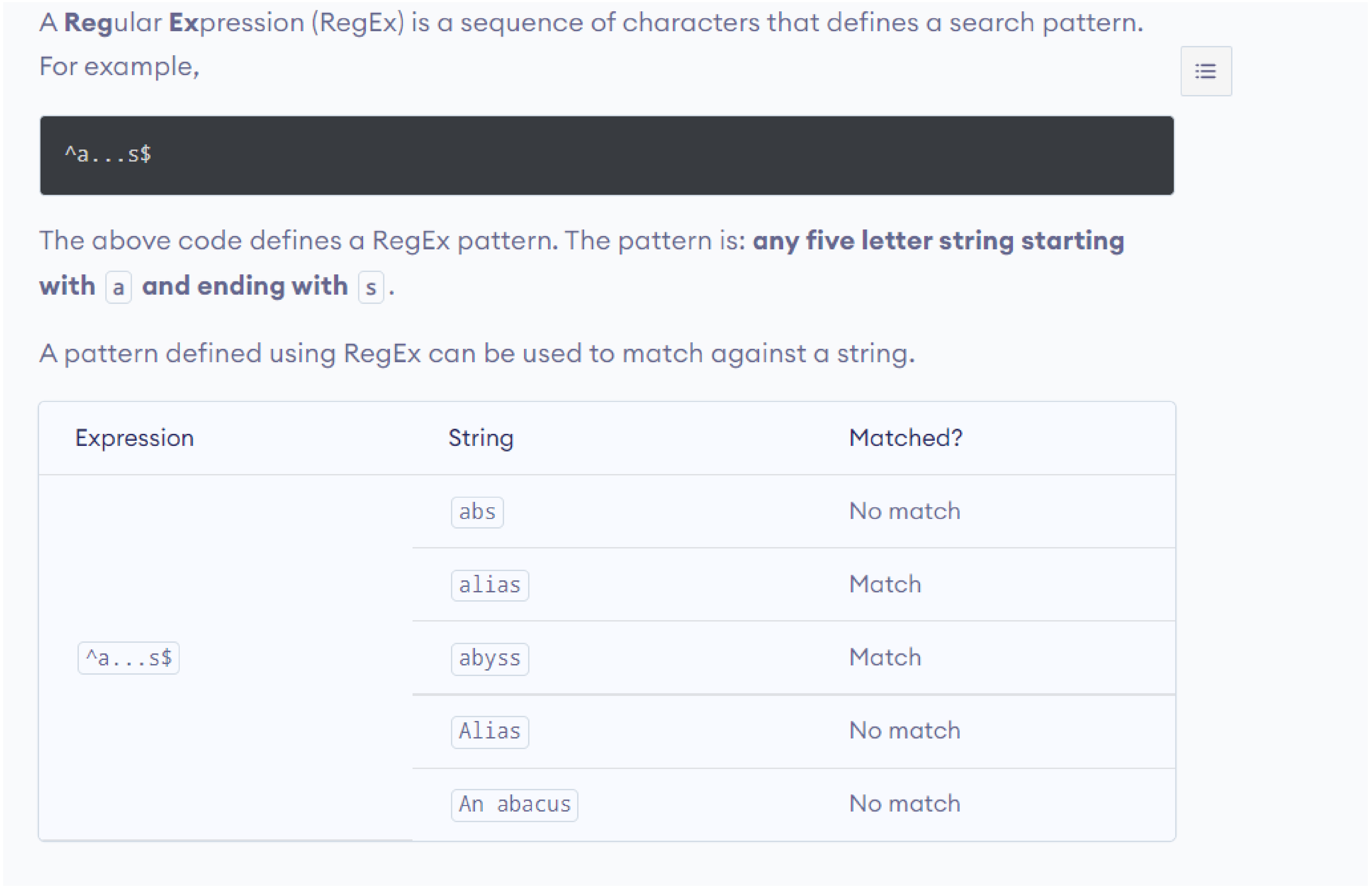
The cache is automatically managed by the computer so generally, the **re.purge()** function is not needed. But, if a particular complex program may have a large compilation time, this function may help.



```
In [8]: import re

        string = "This is a random string containing alphanumeric chara3cters."

In [9]: print(re.search("\w+om", string)) #Compiled Object is created and stored to cache memory

        <re.Match object; span=(10, 16), match='random'>

In [10]: re.purge() #Clears Cache and Frees Memory
```

Here, on executing search() function, the engine creates a compiled regex object and stores it in the cache memory.

By using **re.purge(),** we have cleared the cache memory. Clearing cache frequently can give better performance as memory would not remain cluttered with old, useless data.



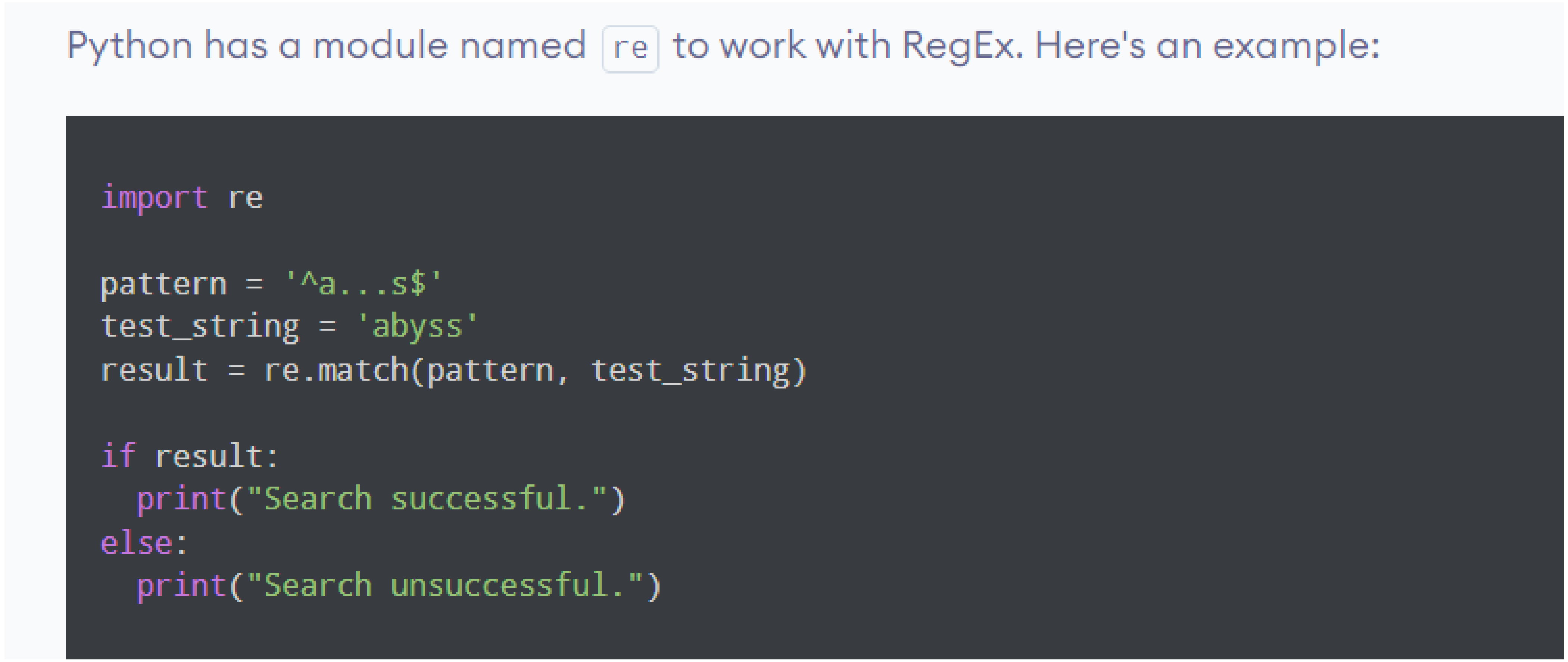
A **Regular Expression** (RegEx) is a sequence of characters that defines a search pattern. For example,

```
^a...s$
```

The above code defines a RegEx pattern. The pattern is: **any five letter string starting with `a` and ending with `s`**.

A pattern defined using RegEx can be used to match against a string.

Expression	String	Matched?
`^a...s$`	`abs`	No match
	`alias`	Match
	`abyss`	Match
	`Alias`	No match
	`An abacus`	No match



Python has a module named `re` to work with RegEx. Here's an example:

```python
import re

pattern = '^a...s$'
test_string = 'abyss'
result = re.match(pattern, test_string)

if result:
  print("Search successful.")
else:
  print("Search unsuccessful.")
```

Here, we used re.match() function to search pattern within the test_string.
The method returns a match object if the search is successful. If not, it returns None.

MetaCharacters

Metacharacters are characters that are interpreted in a special way by a RegEx engine. Here's a list of metacharacters:

[] . ^ $ * + ? {} () \ |

`[]` - Square brackets

Square brackets specifies a set of characters you wish to match.

Expression	String	Matched?
`[abc]`	`a`	1 match
	`ac`	2 matches
	`Hey Jude`	No match
	`abc de ca`	5 matches

Here, `[abc]` will match if the string you are trying to match contains any of the `a`, `b` or `c`.

You can also specify a range of characters using `-` inside square brackets.

- `[a-e]` is the same as `[abcde]`.
- `[1-4]` is the same as `[1234]`.
- `[0-39]` is the same as `[01239]`.

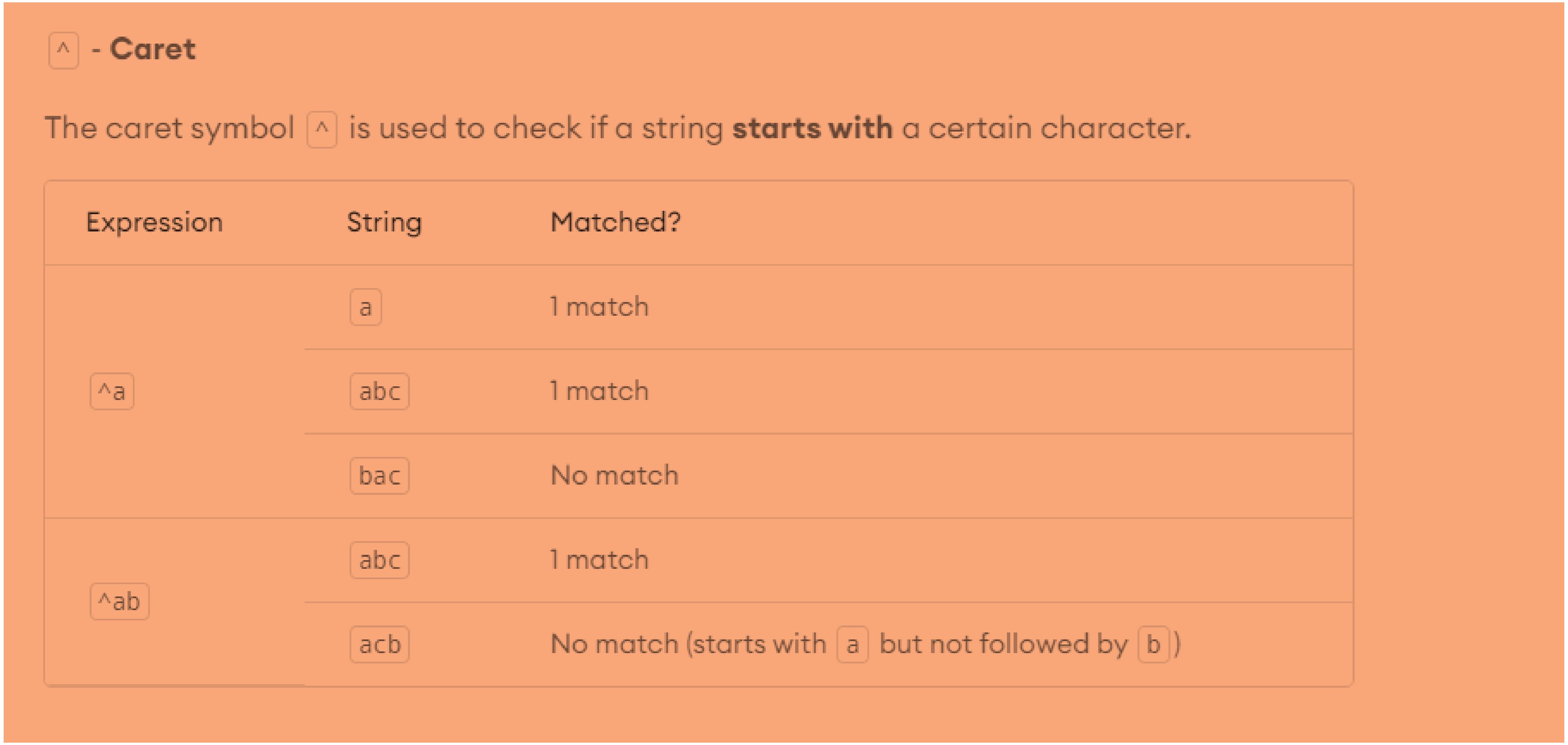
You can complement (invert) the character set by using caret `^` symbol at the start of a square-bracket.

- `[^abc]` means any character except `a` or `b` or `c`.
- `[^0-9]` means any non-digit character.

`.` - **Period**

A period matches any single character (except newline `'\n'`).

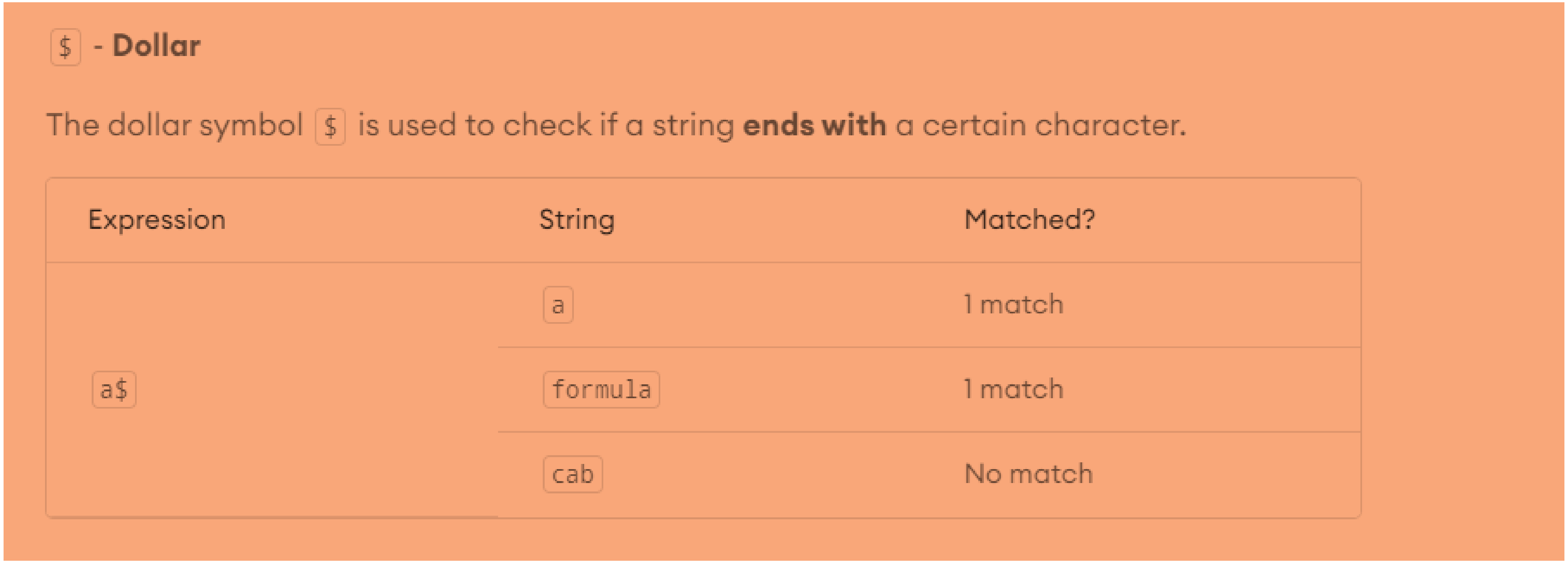
Expression	String	Matched?
`..`	`a`	No match
	`ac`	1 match
	`acd`	1 match
	`acde`	2 matches (contains 4 characters)



`^` - Caret

The caret symbol `^` is used to check if a string **starts with** a certain character.

Expression	String	Matched?
`^a`	`a`	1 match
	`abc`	1 match
	`bac`	No match
`^ab`	`abc`	1 match
	`acb`	No match (starts with `a` but not followed by `b`)



`$` - Dollar

The dollar symbol `$` is used to check if a string **ends with** a certain character.

Expression	String	Matched?
`a$`	`a`	1 match
	`formula`	1 match
	`cab`	No match

`*` - Star

The star symbol `*` matches **zero or more occurrences** of the pattern left to it.

Expression	String	Matched?
`ma*n`	`mn`	1 match
	`man`	1 match
	`maaan`	1 match
	`main`	No match (`a` is not followed by `n`)
	`woman`	1 match

`+` - Plus

The plus symbol `+` matches **one or more occurrences** of the pattern left to it.

Expression	String	Matched?
`ma+n`	`mn`	No match (no `a` character)
	`man`	1 match
	`maaan`	1 match
	`main`	No match (a is not followed by n)
	`woman`	1 match

`?` - Question Mark

The question mark symbol `?` matches **zero or one occurrence** of the pattern left to it.

Expression	String	Matched?
	`mn`	1 match
	`man`	1 match
`ma?n`	`maaan`	No match (more than one `a` character)
	`main`	No match (a is not followed by n)
	`woman`	1 match

`{}` **- Braces**

Consider this code: `{n,m}`. This means at least `n`, and at most `m` repetitions of the pattern left to it.

Expression	String	Matched?
`a{2,3}`	`abc dat`	No match
	`abc daat`	1 match (at `daat`)
	`aabc daaat`	2 matches (at `aabc` and `daaat`)
	`aabc daaaat`	2 matches (at `aabc` and `daaaat`)

Let's try one more example. This RegEx `[0-9]{2, 4}` matches at least 2 digits but not more than 4 digits

Expression	String	Matched?
`[0-9]{2,4}`	`ab123csde`	1 match (match at `ab123csde`)
	`12 and 345673`	3 matches (`12`, `3456`, `73`)
	`1 and 2`	No match

`|` - Alternation

Vertical bar `|` is used for alternation (`or` operator).

Expression	String	Matched?
	`cde`	No match
`a|b`	`ade`	1 match (match at `ade`)
	`acdbea`	3 matches (at `acdbea`)

Here, `a|b` match any string that contains either `a` or `b`

`()` - Group

Parentheses `()` is used to group sub-patterns. For example, `(a|b|c)xz` match any string that matches either `a` or `b` or `c` followed by `xz`

Expression	String	Matched?
	`ab xz`	No match
`(a|b|c)xz`	`abxz`	1 match (match at `abxz`)
	`axz cabxz`	2 matches (at `axzbc cabxz`)

`\` **- Backslash**

Backlash `\` is used to escape various characters including all metacharacters. For example,

`\$a` match if a string contains `$` followed by `a`. Here, `$` is not interpreted by a RegEx engine in a special way.

If you are unsure if a character has special meaning or not, you can put `\` in front of it. This makes sure the character is not treated in a special way.