MAKE
SCHOOL

REGULAR EXPRESSIONS

what the [^#_0-9]!!?

IT'S ALL ABOUT PATTERNS

Regular Expressions work with strings

Strings are sequences of characters (bytes)

Files are sequences of bytes too

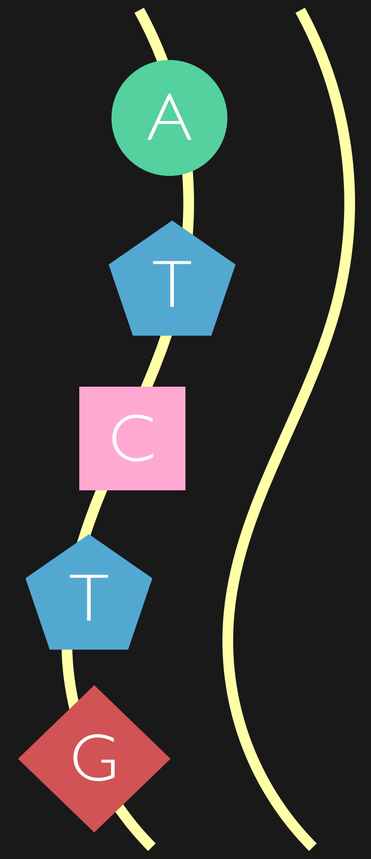
When you have sequences, you have the possibility for patterns

EXAMPLE: DNA ANALYSIS

DNA is a sequence of molecules (A, C, T, and G)

Genes are *patterns* of molecules

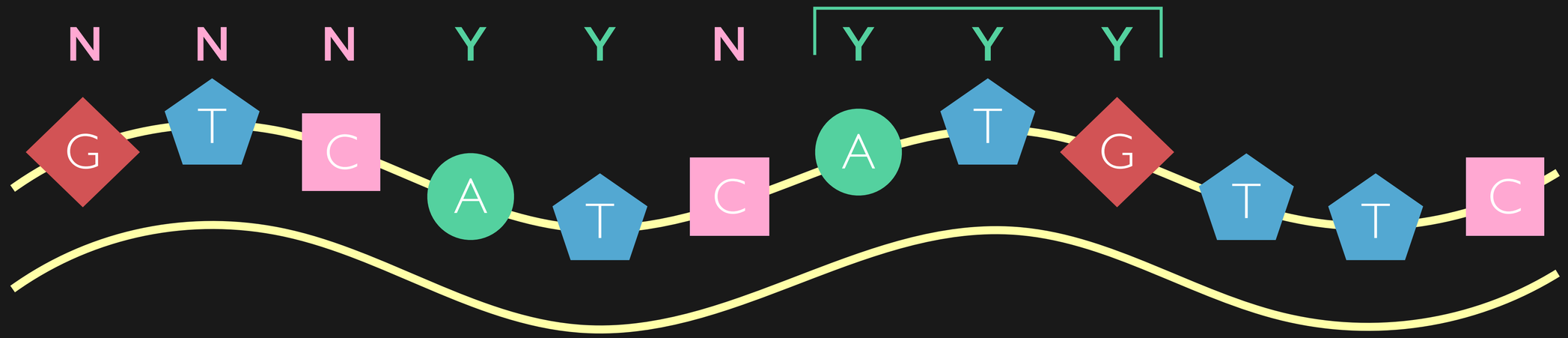
We can search for gene patterns within DNA strands



EXAMPLE: DNA ANALYSIS

Pattern to search for: **ATG**

Start from left, search each base until the pattern matches



MAKE SCHOOL

MATCHING PATTERNS

With a pattern specified, new questions arise:

Is this pattern found in the DNA sample?

How many times does it occur?

At what position does it occur?

What genes come immediately before and after?

EXACT VS. INEXACT

Exact matches: describe a *literal string* to match on (e.g. find command in a text editor)

Inexact matches: describe a *pattern* to match on (what regular expressions do!)

EXAMPLE: FIND QUOTES

How would you describe a pattern to match *all words* enclosed in quotes?

```
regular “expressions” are “super” cool.
```

EXAMPLE: FIND QUOTES

We need to describe a pattern *in the abstract*, to say “match any sequence of characters that starts and ends with a quote character”.

```
regular “expressions” are “super” cool.
```

LET’S SOLVE A PROBLEM

FIND THE PRICES

Here's our problem. We have a breakfast menu, and we need to parse out just the prices.

```
Coffee: $1.75,
Blueberry Scone:
$2.50, 2 Buttermilk
Pancakes: $5,
Oatmeal: $4.25
```

FIND THE PRICES

For problems like this, it's best to work from the simple/general to the complicated/specific.

Let's start by with a regular expression to match all the $ symbols. (Notice the escape character - $ is a special character in regular expressions).

```
pattern = '\$'
```

```
Coffee: $1.75,
Blueberry Scone: $2.50,
2 Buttermilk Pancakes:
$5, Oatmeal: $4.25
```

MAKE SCHOOL

FIND THE PRICES

Now we can refine the pattern by also matching the four characters following a $ sign.

But that is giving some bad data, so...

```
pattern = '\$.{4}'
```

```
Coffee: $1.75,
Blueberry Scone: $2.50,
2 Buttermilk Pancakes:
$5, Oatmeal: $4.25
```

FIND THE PRICES

Let's be more specific: only match *digits* following a $ symbol. Also match an optional decimal.

Better! But we don't want the $ in our final match, just the numbers.

```
pattern =
'\$[0-9]+\.?[0-9]*'

Coffee: $1.75,
Blueberry Scone: $2.50,
2 Buttermilk Pancakes:
$5, Oatmeal: $4.25
```

FIND THE PRICES

Revise the pattern to just match the digits (`\d` is shorthand for `[0-9]`), not the $ symbol.

Well, now we have this extra 2 in there, and that's not a price.

```
pattern =
'\d+\.?\d*'


Coffee: $1.75,
Blueberry Scone: $2.50,
2 Buttermilk Pancakes:
$5, Oatmeal: $4.25
```

FIND THE PRICES

The solution is to use *grouping* to isolate one part of the match.

That's it! Pattern found.

```
pattern =
'\$(\d+\.?\d*)'

Coffee: $1.75,
Blueberry Scone: $2.50,
2 Buttermilk Pancakes:
$5, Oatmeal: $4.25
```

NOT JUST IN PYTHON

Most programming languages have some kind of regular expression feature.

Syntax may differ slightly, but the concepts are the same.

WATCH OUT FOR

Special characters

$, ^, etc. have special meaning, and you need to escape them if you want a literal match.

Flags*

Change the behavior of regular expressions.

*https://docs.python.org/3/library/re.html#module-contents

MAKE SCHOOL

NOW GO PLAY

https://docs.python.org/3/howto/regex.html

https://docs.python.org/3/library/re.html

http://www.diveintopython3.net/regular-expressions.html

Regex Visualizer: http://regexper.com/

MAKE
SCHOOL