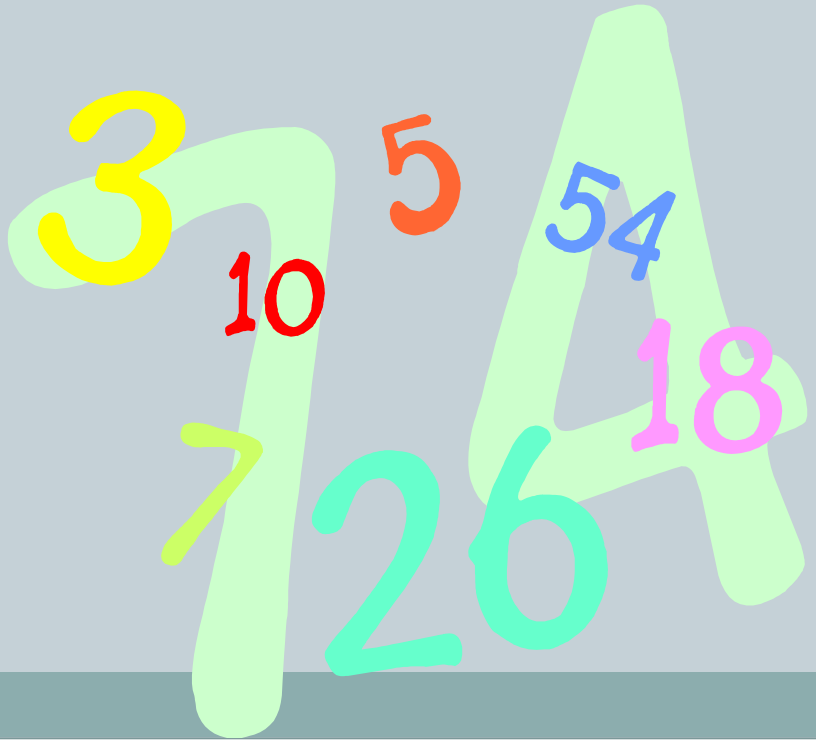
Enumerations

Outline

- Avoiding magic numbers
  - Variables takes on a small set of values
  - Use descriptive names instead of literal values
  - Python enumerations



Variables from a Set of Values

- Magic numbers
  - Where did the value come from?
  - What does it mean?
  - What if you mistype the number?
  - What if you want to keep value in specific range?


```
direction = 0

...

if direction == 1 or direction == 3 or direction == 5 or direction == 7:
    # some code here

direction = 0            // Valid???
direction = 8            // Valid???
direction = -2729        // Valid???
```

Variables from a Set of Values

- Solution 1: Create constants
  - Descriptive names means everybody can read
  - Bugs less likely, typo in name = compile error
  - In Python, could still change value

```
NORTH      = 0
NORTHEAST  = 1
EAST       = 2
SOUTHEAST  = 3
SOUTH      = 4
SOUTHWEST  = 5
WEST       = 6
NORTHWEST  = 7

direction = NORTH

...

if direction == NORTHEAST or direction == SOUTHEAST or /
   direction == SOUTHWEST or direction == NORTHWEST:
    #  some code here
```

MAGIC NUMBERS

Constants not Always Ideal

```
NORTH      = 0
NORTHEAST  = 1
EAST       = 2
SOUTHEAST  = 3
SOUTH      = 4
SOUTHWEST  = 5
WEST       = 6
NORTHWEST  = 7
```

Problem 1: Tedious to type. Also easy to mess up, e.g. setting two constants to same value.

```
direction = 0
...
if direction == NORTHEAST or direction == SOUTHEAST or /
   direction == SOUTHWEST or direction == NORTHWEST:
    #  some code here
```

Problem 2: Not forced to use the friendly names.

```
direction = 0              // Valid???
direction = 8              // Valid???
direction = -2729          // Valid???
```

Problem 3: Not forced to stay in range. What does it mean to be 8 or -2729 if you are a compass direction?

Enumerations

- A better solution: enumerations
  - Specifies exact set of friendly names
  - Compiler ensures we work with compass directions (or whatever enumeration) when we expect to

```
from enum import Enum, auto

class Compass(Enum):
    NORTH = auto()
    NORTHEAST = auto()
    EAST = auto()
    SOUTHEAST  = auto()
    SOUTH = auto()
    SOUTHWEST = auto()
    WEST = auto()
    NORTHWEST = auto()
```

Enumeration Tricks

- Enumerations
  - Actually objects with a few handy methods and attributes:

`name`	Print out friendly name corresponding to value of variable
`value`	Returns the value of the enumerated type member

for-each loop, goes over all values of the enumeration

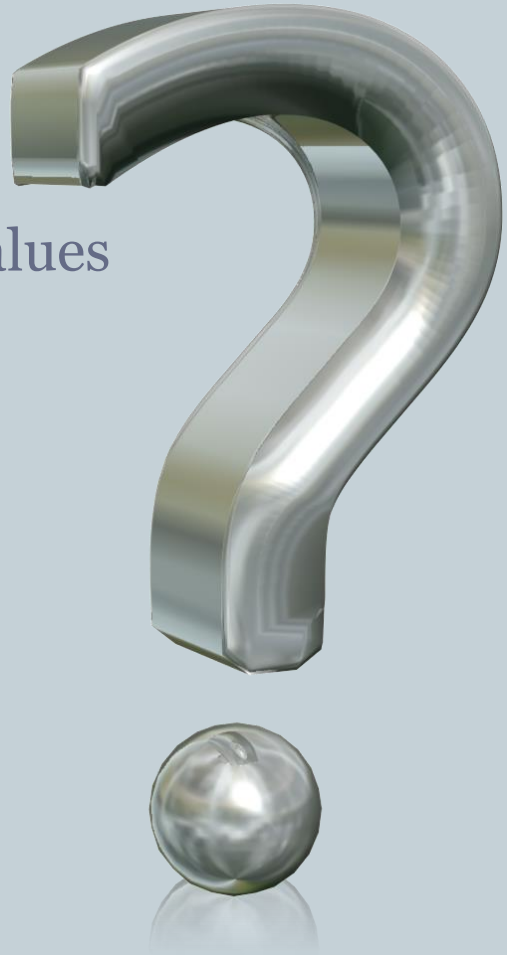
```
from enum import Enum, auto

class Compass(Enum):
    NORTH = auto()
    NORTHEAST = auto()
    EAST = auto()
    SOUTHEAST  = auto()
    SOUTH = auto()
    SOUTHWEST = auto()
    WEST = auto()
    NORTHWEST = auto()

for d in Compass:
    print(d.value)
    print(d.name)
```

Summary

- Avoiding magic numbers
  - Variables takes on a small set of values
  - Use descriptive names instead of literal values
  - Python enumerations



Regular Expressions

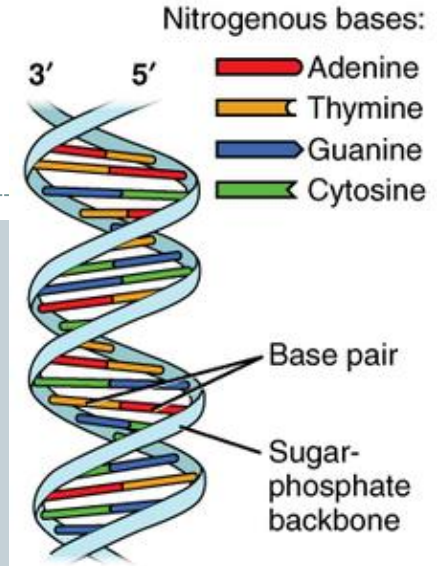
having nothing driving regulating growing pausing bringing stepping knocking not
hing surprising leaning looking striving pacing Nothing loitering falling enchan
ting reaching overlapping receiving meaning going something something taking goi
ng being broiling thing putting lording making anything knowing paying paying be
ing paying being considering having whaling going wh
ling being performing cajoling resulting discriminat
earning reachi
owing shoulder
thing flying h
ng creaking lo
ting chatterin
ng straggling
ing painting s
sweeping deat
ting dark-look
uminating ador
oking nothing
leeping making
ying dusting p
g standing loo

TTAAAGCTGGCGCGGAGGCGGCTGGCGCGGAGGCTG

Outline

- Regular expressions
  - Convenient notation to detect if a string is in a set
    - Built-in to many modern programming languages
    - Usually easier than writing custom string parsing code
  - Very powerful
    - But still some things it can't do:
      - e.g. Recognize all bit strings with equal number of 0's and 1's
  - Well-supported in Python:
    - Test if a string matches an RE
    - Split a string based on an RE
    - Find-and-replace based on an RE

Pattern Matching



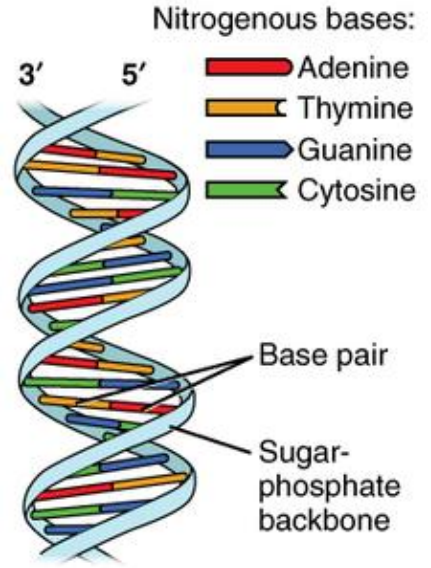
- Is a given string in a set of strings?
  - Example from genomics:
    - DNA: sequence of nucleotides: C, G, A or T
    - Fragile X syndrome:
      - Common cause of mental disability
      - Human genome contains triplet repeats of CGG or AGG, bracketed by GCG at the beginning and CTG at the end
      - Number of repeats is variable, correlated with syndrome

Set of strings: "all strings of G, C, T, A having some occurrence of GCG followed by any number of CGG or AGG triplets, followed by CTG"

Question: Is the following string in this set of strings?

GCGGCGTGTGTGCGAGAGAGTGGGTTTAAAGCTGGCGCGGAGGCGGCTGGCGCGGAGGCTG

Pattern Matching



- Is a given string in a set of strings?
  - Example from genomics:
    - DNA: sequence of nucleotides: C, G, A or T
    - Fragile X syndrome:
      - Common cause of mental disability
      - Human genome contains triplet repeats of CGG or AGG, bracketed by GCG at the beginning and CTG at the end
      - Number of repeats is variable, correlated with syndrome

Set of strings: "all strings of G, C, T, A having some occurrence of GCG followed by any number of CGG or AGG triplets, followed by CTG"

Question: Is the following string in this set of strings?

GCGGCGTGTGTGCGAGAGAGTGGGTTTAAAGCTGGCGCGGAGGCGGCTGGCGCGGAGGCTG

Answer: **Yes**

A Pattern Matching Application

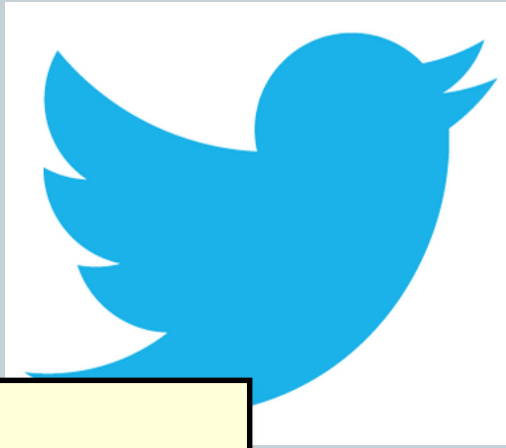
- PROSITE
  - Huge database of protein families and domains
  - How to identify the C_2H_2-type zinc finger domain?
    1. C
    2. Between 2 and 4 amino acids
    3. C
    4. 3 amino acids
    5. One of the following amino acids: LIVMFYWCX
    6. 8 amino acids
    7. H
    8. Between 3 and 5 amino acids
    9. H

CAASCGGPYACGGWAGYHAGWH

CAASCGGPYACGGWAGYHAGWH

Another Pattern Matching Application

- What are people saying about Keith on twitter?
  - Collecting ~1% of tweets since 2010
    - Currently ~~737 GB~~ 1.6 TB compressed!
  - Find all tweets starting with "keith is"
  - How many?
    - Out of 54 M "sensible" English tweets: 91



```
keith is so awesome
keith is fun
keith is beautiful
keith is sweet
keith is the king of this here compound
keith is great
keith is always there when i need to laugh
keith is the bestest
keith is awesome
keith is so sweet
keith is hilarious
keith is such a kind soul and life saver
...
```

Even More Applications

- Test if a string matches some pattern
  - Process natural language
  - Scan for virus signatures
  - Access information in digital libraries
  - Find-and-replace in word processors
  - Filter text (spam, NetNanny, ads, Carnivore, malware)
  - Validate text fields (dates, email, URL, credit card)
- Parse text files
  - Compile a Python program
  - Crawl and index the web
  - Create Python documentation from comments

Regular Expressions

- Regular expressions (REs)
  - Notation that specifies a set of strings

operation	regular expression	matches	does not match
concatenation	`aabaab`	`aabaab`	*every other string*
wildcard `.`	`.u.u.u.`	`cumulus` `jugulum`	`succubus` `tumultuous`
union `|`	`(aa)|(baab)`	`aa` `baab`	*every other string*
closure / star (0 or more) `*`	`ab*a`	`aa` `abbba`	`ab` `ababa`
parentheses `()`	`a(a|b)aab`	`aaaab` `abaab`	*every other string*
	`(ab)*a`	`a` `abababababa`	`aa` `abbba`

Regular Expressions

- Regular expressions (REs)
  - Notation is surprisingly expressive

regular expression	matches	does not match
`.*spb.*` *contains the trigraph spb*	`raspberry` `crispbread`	`subspace` `subspecies`

`a* | (a*ba*ba*ba*)*`
multiple of three b's
	`bbb`
	`aaa`
	`bbbaababbaa`
		`b`
		`bb`
		`baabbbaa`

`.*0....` *fifth to last digit is 0*	`1000234` `98701234`	`111111111` `403982772`

`gcg(cgg|agg)*ctg`
fragile X syndrome indicator
	`gcgctg`
	`gcgcggctg`
	`gcgcggaggctg`
		`gcgcgg`
		`cggcggcggctg`
		`gcgcaggctg`

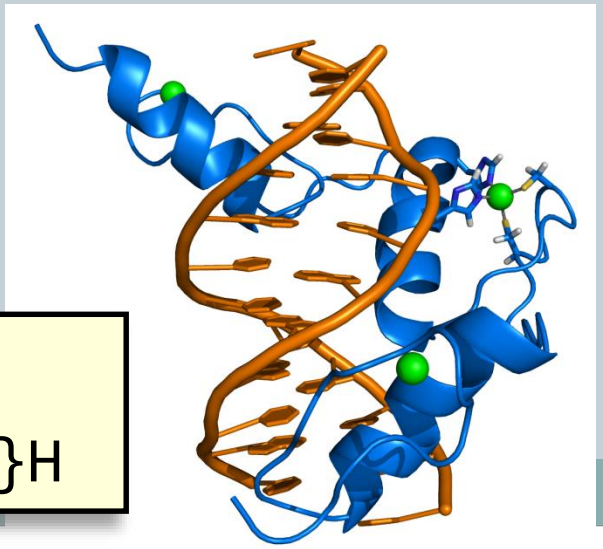
Regular Expressions

- Regular expressions (REs)
  - A standard programmer's tool
    - Built into many languages: Java, Perl, Unix, Python, ...
  - Additional convenience operations:
    - e.g. `[a-e]+` shorthand for `(a|b|c|d|e)(a|b|c|d|e)*`
    - e.g. `\s` is shorthand for any whitespace character

operation	regular expression	matches	does not match
one or more `+`	`a(bc)+de`	`abcde` `abcbcde`	`ade` `bcde`
character class `[]`	`[A-Za-z][a-z]*`	`lowercase` `Capitalized`	`camelCase` `4illegal`
exactly k, between k and j `{k}, {k,j}`	`[0-9]{5}-[0-9]{4}`	`08540-1321` `19072-5541`	`111111111` `166-54-1111`
negation `^`	`[^aeiou]{5,6}`	`rhythm` `synch`	`decade` `rhythms`

Pattern Matching Application

- PROSITE
  - Huge database of protein families and domains
  - Identify the C_2H_2-type zinc finger domain, how???
    1. C
    2. Between 2 and 4 amino acids
    3. C
    4. 3 more amino acids
    5. One of the following amino acids: LIVMFYWCX
    6. 8 more amino acids
    7. H
    8. Between 3 and 5 more amino acids
    9. H



Use a regular expression!

```
C.{2,4}C...[LIVMFYWC].{8}H.{3,5}H
```

Regular Expressions in Python

- Helps match and split up strings
  - Built-in to Python re class methods
  - Note: escape \ in regular expression with \\

Method	Pattern	String	Result
match	“ish”	fishbowl ishmael	no match match
fullmatch	“ish”	fishbowl ishmael	no match no match
sub (all)	“ish”	fishbowl ishmael	fowlbowl owlmael
sub (first)	“ish”	I wish I was a fish	I wowl I was a fish
split	“ish”	I wish I was a fish	[‘I w’, ‘ I was a f’, ‘’]

Regular Expression Example

- Goal: Display all words in a file ending -ing

```
% python GerundFinder mobydick.txt .+ing

having nothing driving regulating growing pausing bringing stepping knocking not
hing surprising leaning looking striving pacing Nothing loitering falling enchan
ting reaching overlapping receiving meaning going something something taking goi
ng being broiling thing putting lording making anything knowing paying paying be
ing paying being considering having whaling going whaling something "Whaling wha
ling being performing cajoling resulting discriminating overwhelming attending e
verlasting ignoring whaling Quitting learning reaching following whaling somethi
ng everything monopolizing having following shouldering comparing halting pausin
g tinkling stopping moving proceeding thing flying hearing sitting beating weepi
ng wailing teeth-gnashing backing Moving creaking looking swinging painting repr
esenting swinging leaning howling toasting chattering shaking everlasting making
 holding being blubbering going Entering straggling reminding painting understan
ding throwing something hovering floating painting something weltering purposing
 spring impaling glittering resembling sweeping death-harvesting horrifying whal
ing sojourning Crossing howling Projecting dark-looking goggling cheating enteri
ng examining telling tapping sharing ruminating adorning stooping working trying
 adjoining Nothing winding scalding looking nothing knowing evening rioting Star
ting offing tramping capering making sleeping making dazzling seeming sleeping s
leeping being getting going feeling saying dusting planing grinning spraining pl
aning gathering throwing yoking leaving standing looking seeing spending cherish
```

GerundFinder

```
% python GerundFinder mobydick.txt .+ing
```

```
import sys
import re

fileName = sys.argv[1]
pattern = sys.argv[2]
p = re.compile(pattern)
with open(fileName) as f:
        words = f.read().split()
for word in words:
        if not p.match(word) == None:
                print(p.match(word).group())
```

Get the name of the file

Get the pattern: ".+ing"

Regular Expression Quick Reference

Construct	Matches
`.`	Any character
`\d`	A digit: 0-9
`\s`	A whitespace character
`\w`	A word character: a-z A-Z 0-9 _
`\D`	A non-digit (anything except 0-9)
`\S`	A non-whitespace character
`\W`	A non-word character

Classes	Matches
`[abc]`	Character a, b or c
`[^abc]`	Any character except a, b, or c
`[a-z]`	Characters a, b, c, ..., z
`[A-Z]`	Characters A, B, C, ..., Z
`[a-zA-Z]`	Characters a, A, b, B, ..., z, Z

Expression	Example matches
`...`	cat, sat, mat, ...
`c..`	cat, cow, cut, ...
`[abc]at`	aat, bat, cat
`[abc]+z`	az, bz, cz, aaz, abz, bcz, bbacz, ...
`[0-9]{5}`	12345, 59701, 01234, ...
`\d\d\d\d`	1980, 2005, 9999, ...

Quantifier	Matches
`*`	Zero or more occurrences
`+`	One or more occurrences
`?`	Zero or one occurrences
`{n}`	Exactly n occurrences
`{n,}`	At least n occurrences
`{n,m}`	Between n and m occurrences inclusive

Summary

- Regular expressions
  - Convenient notation to detect if a string is in a set
    - Built-in to many modern programming languages
    - Usually easier than writing custom string parsing code
  - Very powerful
    - But still some things it can't do:
      - e.g. Recognize all bit strings with equal number of 0's and 1's
  - Well-supported in Python:
    - Test if a string matches an RE
    - Split a string based on an RE
    - Find-and-replace based on an RE