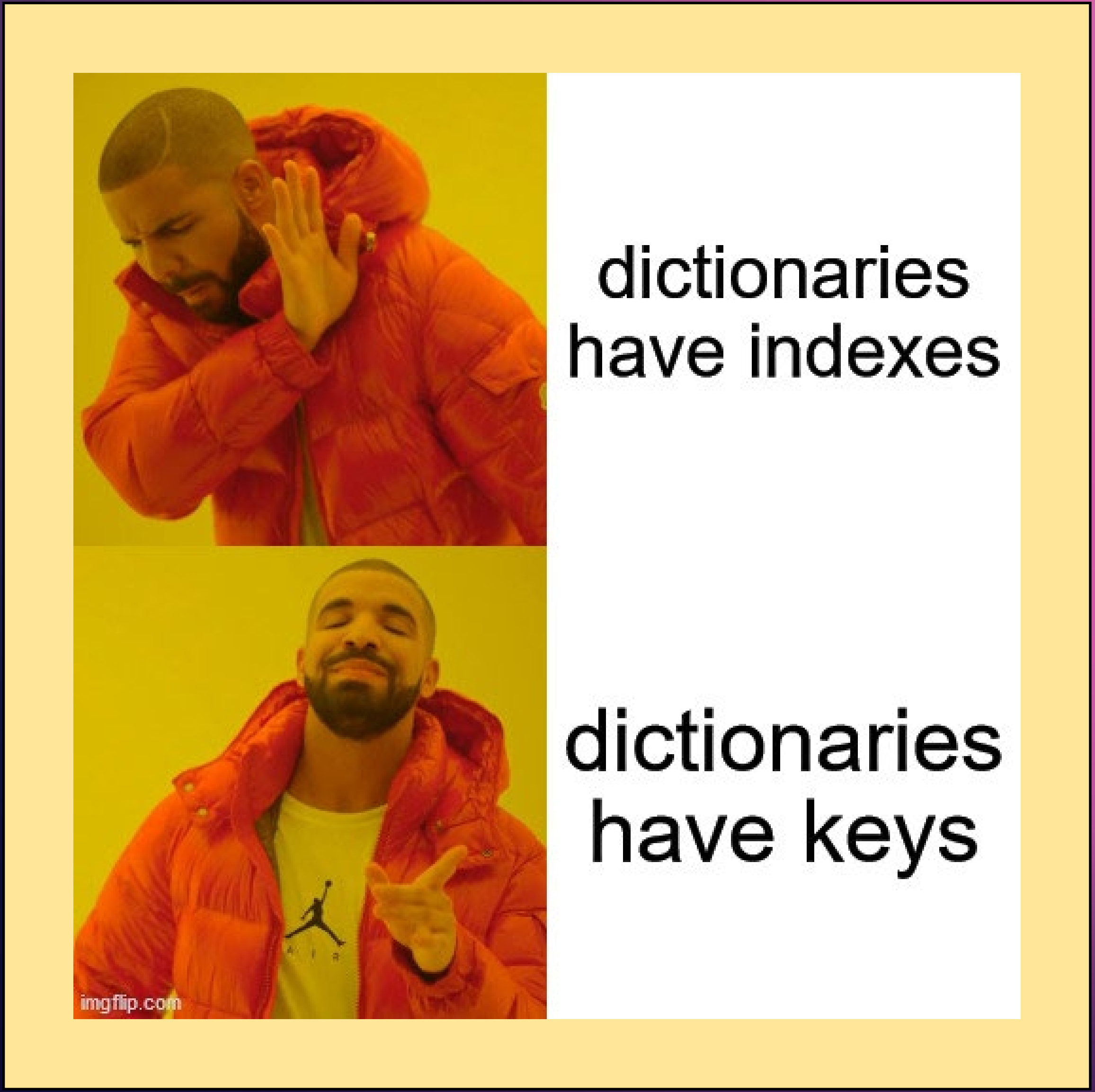
CMPSC 100 SPRING 2021

Look it up

“curly” braces

```
a_dictionary = {}
```


dictionaries have indexes
dictionaries have keys
imgflip.com

Dictionaries:

- Are “associative”
  - Have key:value “pairs”
- Use “curly” braces
- Do not use “indexes”
  - All references are to keys
- Often mix data types

```
fruit = {
      “name”: “apple”,
      “cost”: “.99”,
      “weight”: 1, # That’s a heavy apple?
      “color”: “red”, # Red Delicious r best, no @ me
      “has_worm”: True
}
```

“subscript” notation

```
print(fruit[“name”])
```

```
> apple
```

```
print(fruit[“weight”])

> 1
```

Fruit	Cost	Quantity
Apple	.99	10
Banana	1.10	5
Orange	.75	15

```
fruit_crate = {
    “apple”: {“cost”:.99, “quanitity”: 10},
    “banana”: {“cost”:1.10, “quantity”: 5},
    “orange”: {“cost”: .75, “quantity”: 15}
}
```

```
fruit_crate = {
    “apple”: {“cost”:.99, “quanitity”: 10},
    “banana”: {“cost”:1.10, “quantity”: 5},
    “orange”: {“cost”: .75, “quantity”: 15}
}

print(fruit_crate[“apple”])

> {“cost”: .99, “quantity”: 10}
```

Fruit	Cost	Quantity
Apple	.99	10
Banana	1.10	5
Orange	.75	15

```
print(fruit_crate[“apple”][“cost”])

> .99

print(fruit_crate[“apple”][“quantity”])

> 10
```

```
for fruit in fruit_crate:
    data = fruit_crate[fruit]
    print(data)

> {“cost”: …, “quanitity”: …}
.
.
.
```

And now for something
(not so, but yet kinda)
completely different

0	0	000	NULL		32	20	040	&#032;	Space	64	40	100	&#064;	@	96	60	140	&#096;	`
1	1	001	Start of Header		33	21	041	&#033;	!	65	41	101	&#065;	A	97	61	141	&#097;	a
2	2	002	Start of Text		34	22	042	&#034;	"	66	42	102	&#066;	B	98	62	142	&#098;	b
3	3	003	End of Text		35	23	043	&#035;	#	67	43	103	&#067;	C	99	63	143	&#099;	c
4	4	004	End of Transmission		36	24	044	&#036;	$	68	44	104	&#068;	D	100	64	144	&#100;	d
5	5	005	Enquiry		37	25	045	&#037;	%	69	45	105	&#069;	E	101	65	145	&#101;	e
6	6	006	Acknowledgment		38	26	046	&#038;	&	70	46	106	&#070;	F	102	66	146	&#102;	f
7	7	007	Bell		39	27	047	&#039;	'	71	47	107	&#071;	G	103	67	147	&#103;	g
8	8	010	Backspace		40	28	050	&#040;	(	72	48	110	&#072;	H	104	68	150	&#104;	h
9	9	011	Horizontal Tab		41	29	051	&#041;	)	73	49	111	&#073;	I	105	69	151	&#105;	i
10	A	012	Line feed		42	2A	052	&#042;	*	74	4A	112	&#074;	J	106	6A	152	&#106;	j
11	B	013	Vertical Tab		43	2B	053	&#043;	+	75	4B	113	&#075;	K	107	6B	153	&#107;	k
12	C	014	Form feed		44	2C	054	&#044;	,	76	4C	114	&#076;	L	108	6C	154	&#108;	l
13	D	015	Carriage return		45	2D	055	&#045;	-	77	4D	115	&#077;	M	109	6D	155	&#109;	m
14	E	016	Shift Out		46	2E	056	&#046;	.	78	4E	116	&#078;	N	110	6E	156	&#110;	n
15	F	017	Shift In		47	2F	057	&#047;	/	79	4F	117	&#079;	O	111	6F	157	&#111;	o
16	10	020	Data Link Escape		48	30	060	&#048;	0	80	50	120	&#080;	P	112	70	160	&#112;	p
17	11	021	Device Control 1		49	31	061	&#049;	1	81	51	121	&#081;	Q	113	71	161	&#113;	q
18	12	022	Device Control 2		50	32	062	&#050;	2	82	52	122	&#082;	R	114	72	162	&#114;	r
19	13	023	Device Control 3		51	33	063	&#051;	3	83	53	123	&#083;	S	115	73	163	&#115;	s
20	14	024	Device Control 4		52	34	064	&#052;	4	84	54	124	&#084;	T	116	74	164	&#116;	t
21	15	025	Negative Ack.		53	35	065	&#053;	5	85	55	125	&#085;	U	117	75	165	&#117;	u
22	16	026	Synchronous idle		54	36	066	&#054;	6	86	56	126	&#086;	V	118	76	166	&#118;	v
23	17	027	End of Trans. Block		55	37	067	&#055;	7	87	57	127	&#087;	W	119	77	167	&#119;	w
24	18	030	Cancel		56	38	070	&#056;	8	88	58	130	&#088;	X	120	78	170	&#120;	x
25	19	031	End of Medium		57	39	071	&#057;	9	89	59	131	&#089;	Y	121	79	171	&#121;	y
26	1A	032	Substitute		58	3A	072	&#058;	:	90	5A	132	&#090;	Z	122	7A	172	&#122;	z
27	1B	033	Escape		59	3B	073	&#059;	;	91	5B	133	&#091;	[	123	7B	173	&#123;	{
28	1C	034	File Separator		60	3C	074	&#060;	<	92	5C	134	&#092;	\	124	7C	174	&#124;	|
29	1D	035	Group Separator		61	3D	075	&#061;	=	93	5D	135	&#093;	]	125	7D	175	&#125;	}
30	1E	036	Record Separator		62	3E	076	&#062;	>	94	5E	136	&#094;	^	126	7E	176	&#126;	~
31	1F	037	Unit Separator		63	3F	077	&#063;	?	95	5F	137	&#095;	_	127	7F	177	&#127;	Del

strings are essentially groups
of characters - symbols interpretable
for human convenience.

integer

4 != “4”

string

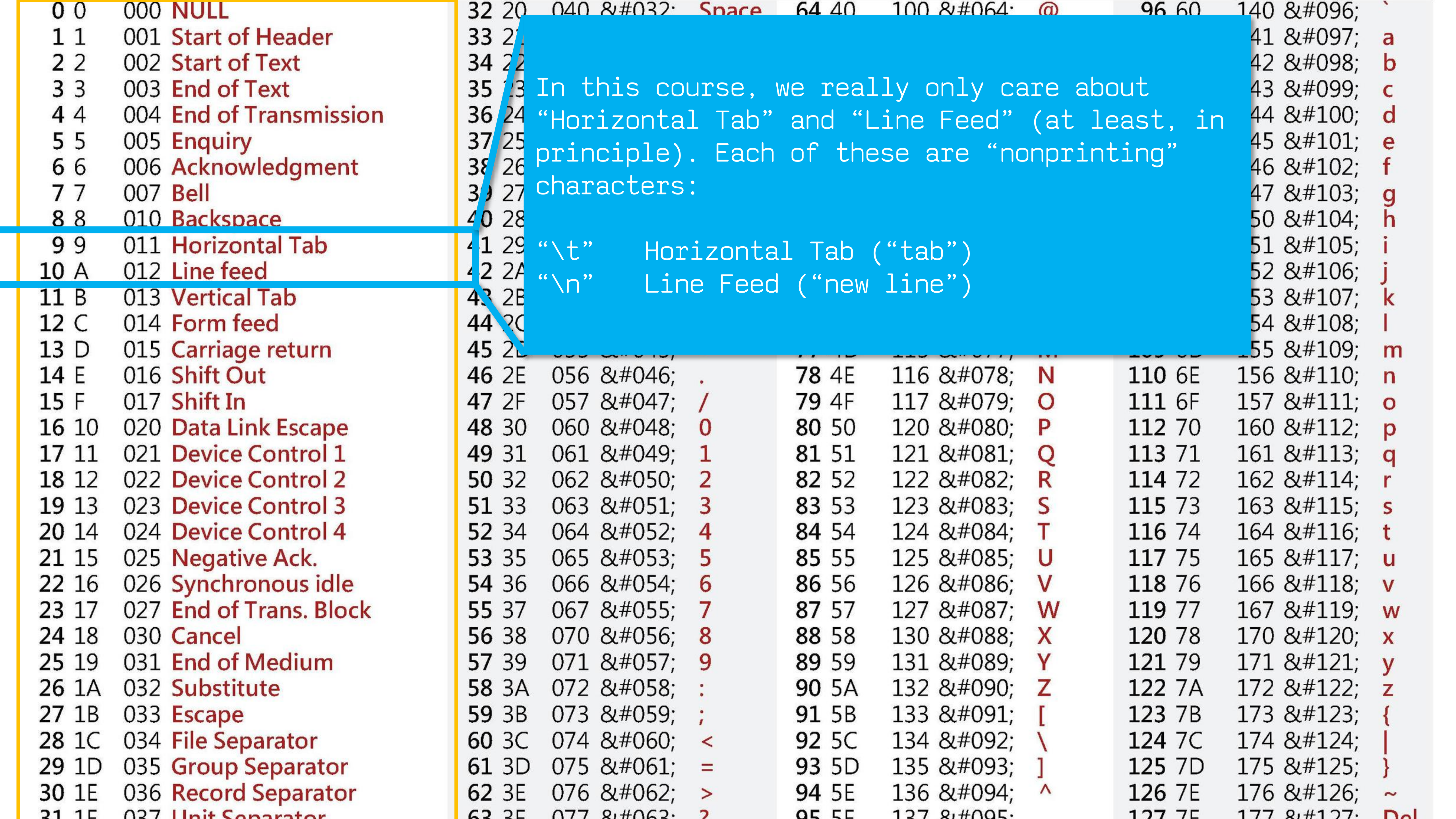
Bottom line:

Computer don’t care

¯_(ツ)_/¯

In this course, we really only care about “Horizontal Tab” and “Line Feed” (at least, in principle). Each of these are “nonprinting” characters:

“\t” Horizontal Tab (“tab”)
“\n” Line Feed (“new line”)



0	0	000	NULL	32	20	040	&#032;	Space	64	40	100	&#064;	@	96	60	140	&#096;	`
1	1	001	Start of Header	33	2												&#097;	a
2	2	002	Start of Text	34	2												&#098;	b
3	3	003	End of Text	35	23												&#099;	c
4	4	004	End of Transmission	36	24												&#100;	d
5	5	005	Enquiry	37	25												&#101;	e
6	6	006	Acknowledgment	38	26												&#102;	f
7	7	007	Bell	39	27												&#103;	g
8	8	010	Backspace	40	28												&#104;	h
9	9	011	Horizontal Tab	41	29												&#105;	i
10	A	012	Line feed	42	2A												&#106;	j
11	B	013	Vertical Tab	43	2B												&#107;	k
12	C	014	Form feed	44	2C												&#108;	l
13	D	015	Carriage return	45	2D												&#109;	m
14	E	016	Shift Out	46	2E	056	&#046;	.	78	4E	116	&#078;	N	110	6E	156	&#110;	n
15	F	017	Shift In	47	2F	057	&#047;	/	79	4F	117	&#079;	O	111	6F	157	&#111;	o
16	10	020	Data Link Escape	48	30	060	&#048;	0	80	50	120	&#080;	P	112	70	160	&#112;	p
17	11	021	Device Control 1	49	31	061	&#049;	1	81	51	121	&#081;	Q	113	71	161	&#113;	q
18	12	022	Device Control 2	50	32	062	&#050;	2	82	52	122	&#082;	R	114	72	162	&#114;	r
19	13	023	Device Control 3	51	33	063	&#051;	3	83	53	123	&#083;	S	115	73	163	&#115;	s
20	14	024	Device Control 4	52	34	064	&#052;	4	84	54	124	&#084;	T	116	74	164	&#116;	t
21	15	025	Negative Ack.	53	35	065	&#053;	5	85	55	125	&#085;	U	117	75	165	&#117;	u
22	16	026	Synchronous idle	54	36	066	&#054;	6	86	56	126	&#086;	V	118	76	166	&#118;	v
23	17	027	End of Trans. Block	55	37	067	&#055;	7	87	57	127	&#087;	W	119	77	167	&#119;	w
24	18	030	Cancel	56	38	070	&#056;	8	88	58	130	&#088;	X	120	78	170	&#120;	x
25	19	031	End of Medium	57	39	071	&#057;	9	89	59	131	&#089;	Y	121	79	171	&#121;	y
26	1A	032	Substitute	58	3A	072	&#058;	:	90	5A	132	&#090;	Z	122	7A	172	&#122;	z
27	1B	033	Escape	59	3B	073	&#059;	;	91	5B	133	&#091;	[	123	7B	173	&#123;	{
28	1C	034	File Separator	60	3C	074	&#060;	<	92	5C	134	&#092;	\	124	7C	174	&#124;	|
29	1D	035	Group Separator	61	3D	075	&#061;	=	93	5D	135	&#093;	]	125	7D	175	&#125;	}
30	1E	036	Record Separator	62	3E	076	&#062;	>	94	5E	136	&#094;	^	126	7E	176	&#126;	~
31	1F	037	Unit Separator	63	3F	077	&#063;	?	95	5F	137	&#095;		127	7F	177	&#127;	Del

Because they're groups
of individual characters,
strings function as a kind
of data structure (particularly
lists)

```
name = “G. Wiz”

for letter in name:
    print(letter)
```

```
>    G
     .

     W
     i
     z
```

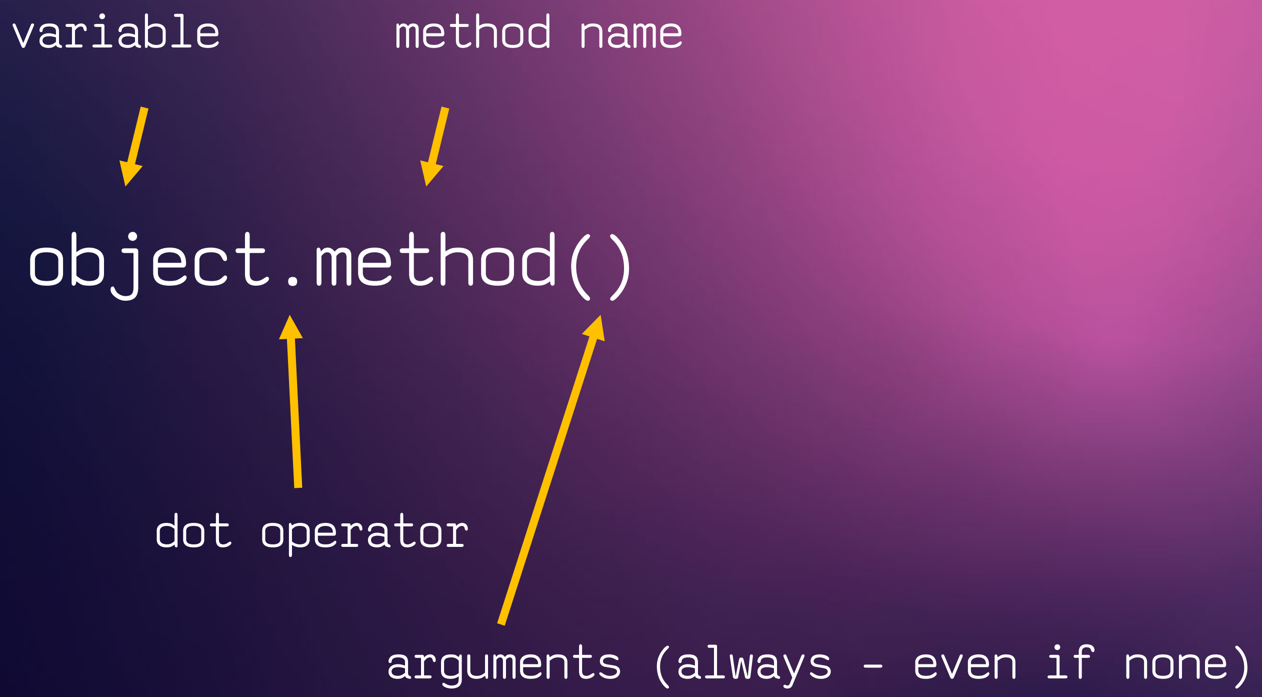
Also like lists:

- Have known length
- Have methods
- Can be sliced

Methods are the “powers” of a given object.

Because objects do different things, methods are often completely different.

Methods always follow dot notation



```
object.method()
```

Various useful methods of string:

- replace
- count
- upper
- lower
- startswith
- endswith

I don’t know them all, and it’s actually somewhat pointless to Memorize all of them.

If we want to know what methods something has, we generally just look it up.

The interwebs

We can also format strings using something called an f-string

We can create “templates” that we can use to print Formatted statements more easily.

Place the variable to print
in curly braces.

```
name = input(“What’s your name? “)
print(f“Oh hello, {name}”)
```

Prefix the string with an f

And, we an also do more than this with strings: we can load them from files using the open function:

```
open(“file.txt”, “r”)
```

File path

File mode

We pull file contents in by using the

read or

readlines

methods.

Reads all file contents as a string

Reads all file contents as a list, separating all lines by \n