
WELCOME!


python

Hi, I'm Colt

I've lead in-person programming bootcamps in the Bay Area for a decade.

I've been a lead-instructor and curriculum director for multiple bootcamps including Galvanize and Rithm School where we guarantee students jobs.

I've taught millions of students to code online and was selected as the Best New Instructor on Udemy!

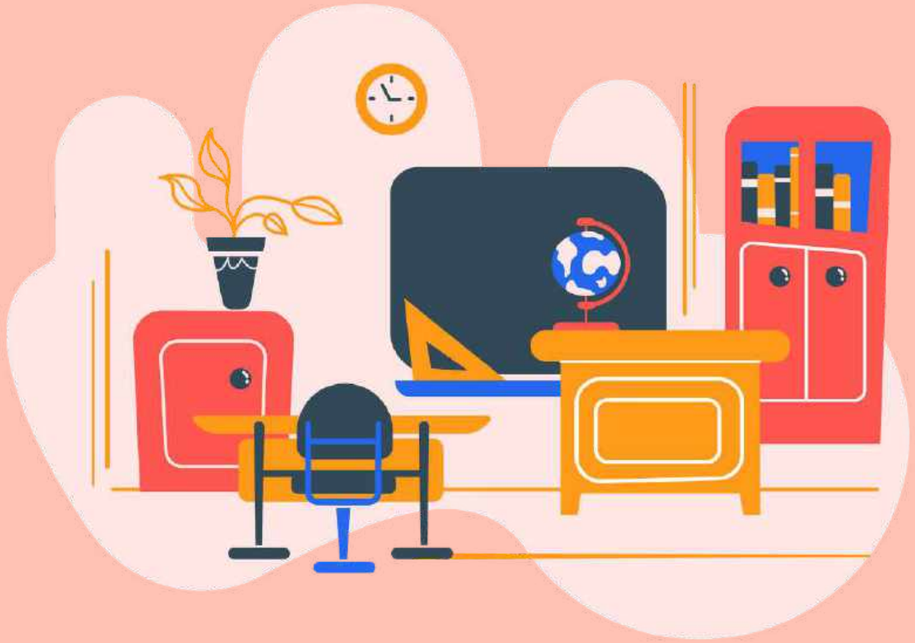
NEVER KNOW WHAT TO PUT UP HERE

My students work at companies including...

amazon

Pinterest

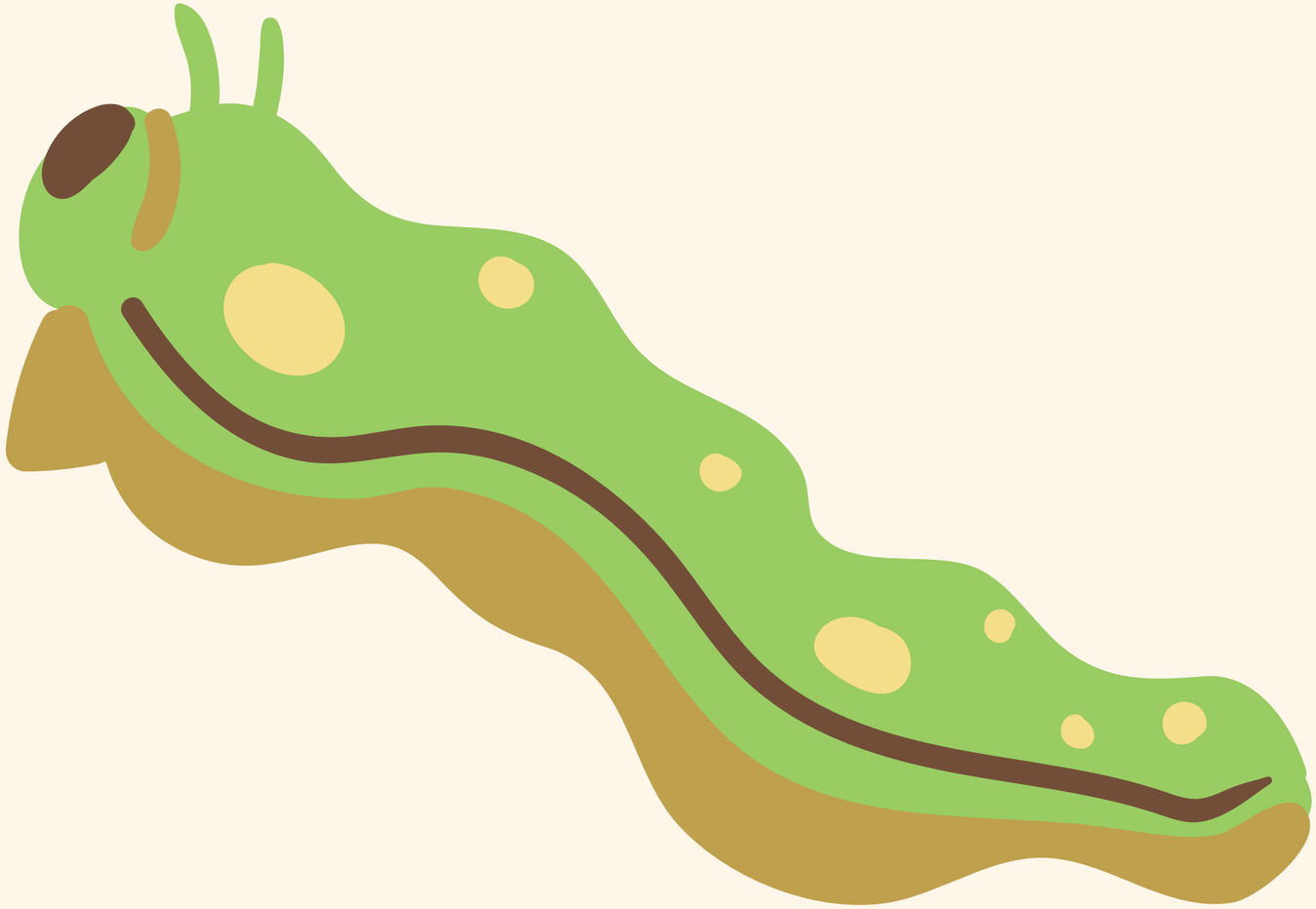
salesforce



What really matters...

Is that I've spent years trying to perfect teaching programming to beginners

I've learned how to keep students engaged from my years of in-person teaching experience.



This Course Is...

- A fast, effective pathway to learn Python
- Full of exercises/challenges/quizzes
- Made for normal people who can't stand 20 minute videos. Average video is 4 minutes.
- An ideal on-ramp to data science, ML, or web development courses.

WHAT THIS COURSE IS NOT

This course isn't

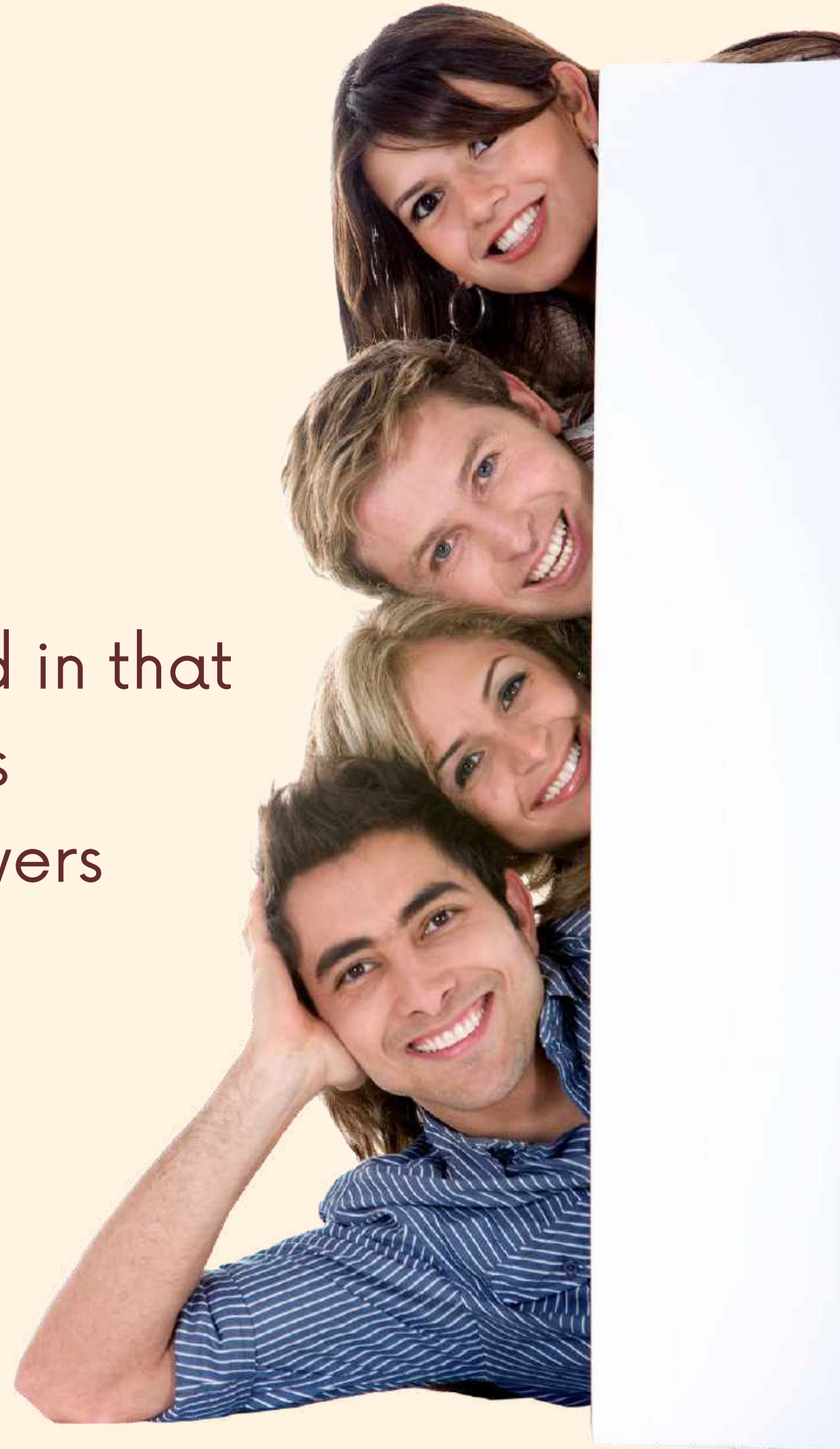
- A 60 hour Python encyclopedia course that covers every possible feature of Python
- An advanced level Python course for experts
- A web development, data science, or machine learning oriented course

High Demand

There are more Python jobs today than ever before, and salaries are at an all-time high. If you could only pick one language to learn, it should be Python.

Huge Community

Python has been around for 30+ years now, and in that time a massive, supportive online community has formed. You can always get help and find answers (relatively) easily.



Libraries

There are thousands of Python libraries, frameworks and tools available. From django to matplotlib to scikitlearn, there's usually something for every need.

Easy To Learn

Python is (in my opinion) the easiest programming language to learn. It has far fewer quirks and oddities than languages like JavaScript.

Versatility

You can use Python for simple scripting tasks.
You can use it to build huge web applications.
You can use it for data processing and analysis.
And of course, you can use it for machine learning!

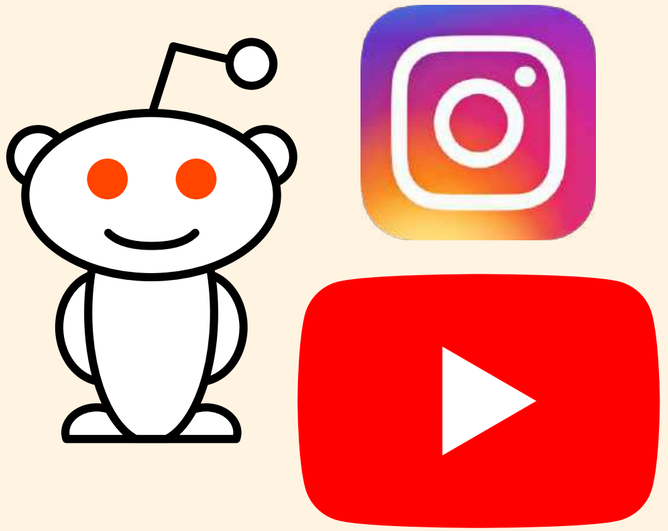
What can we build with python

Game Dev

Hugely popular games including the Sims 4, Civilization, EVE Online, and many others are built using Python!

Web Dev

Companies ranging from tiny startups to massive unicorns use Python to build web applications.



Data Stuff

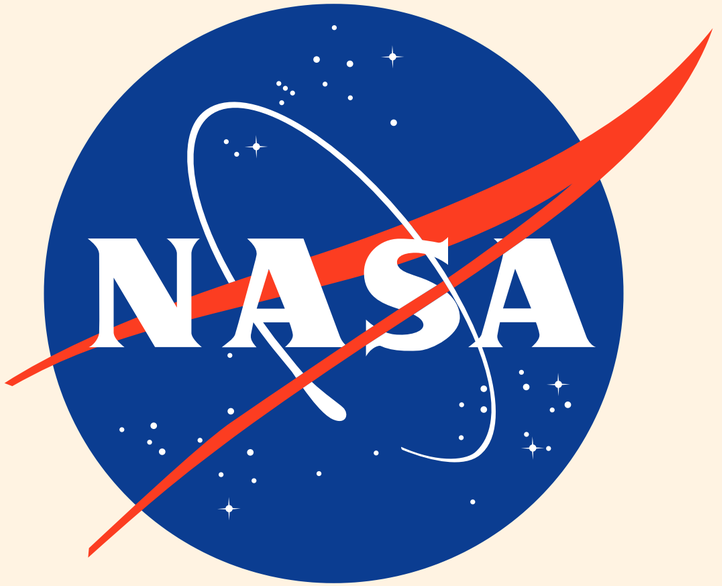
Python is hugely popular in the world of data science and data analysis. Python tools like numpy, pandas, and matplotlib are widely used.

Machine Learning

Python is "the" language for machine learning. Tools including TensorFlow, Scikitlearn, OpenCV, and NLTK require Python knowledge

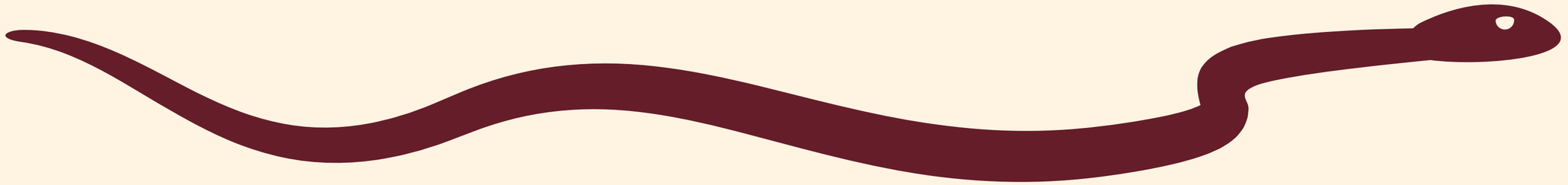
Science

Python is used across academia and the science world, from biology research labs to NASA engineering teams



A Tiny Little Bit Of

Python History



A Tiny Little Bit Of

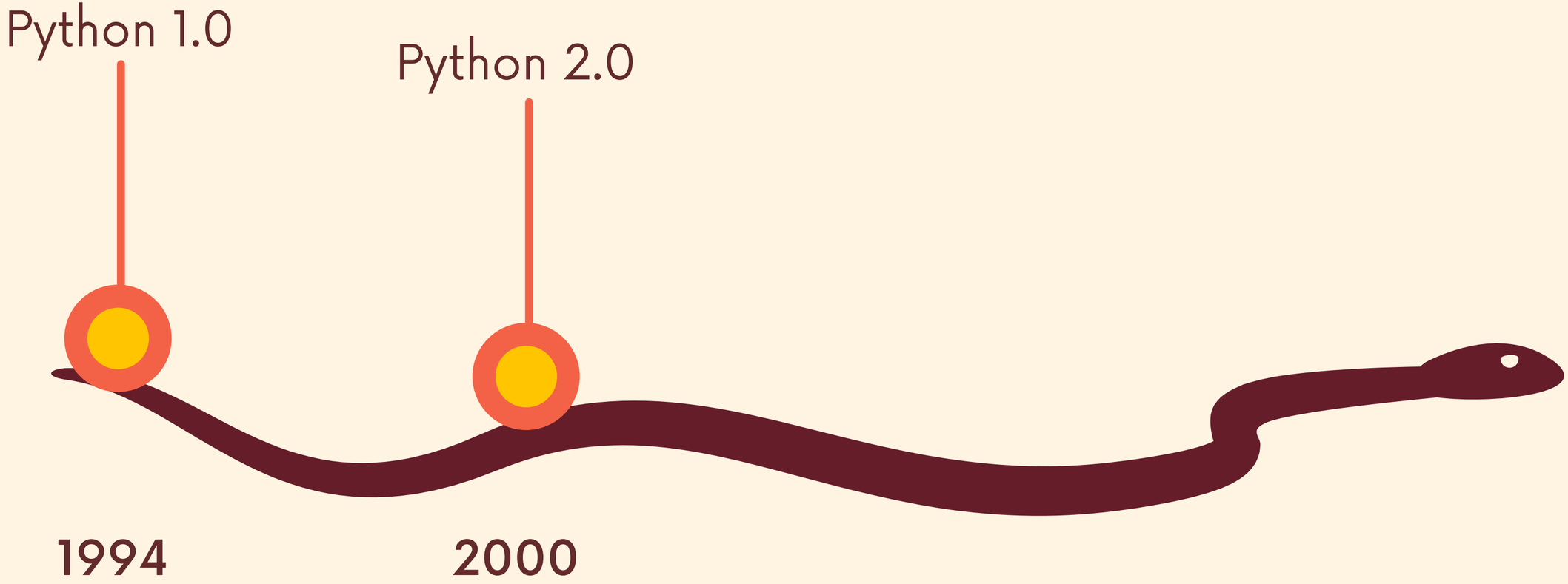
Python History

Python 1.0

1994

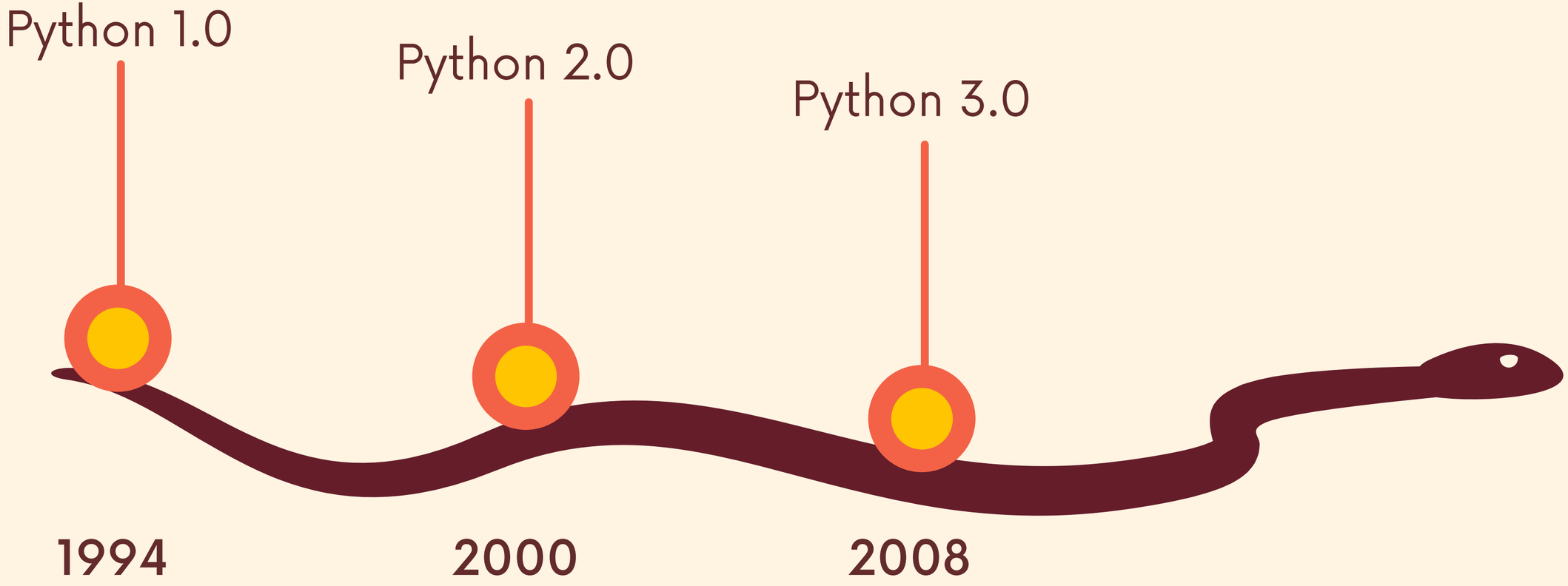
A Tiny Little Bit Of

Python History



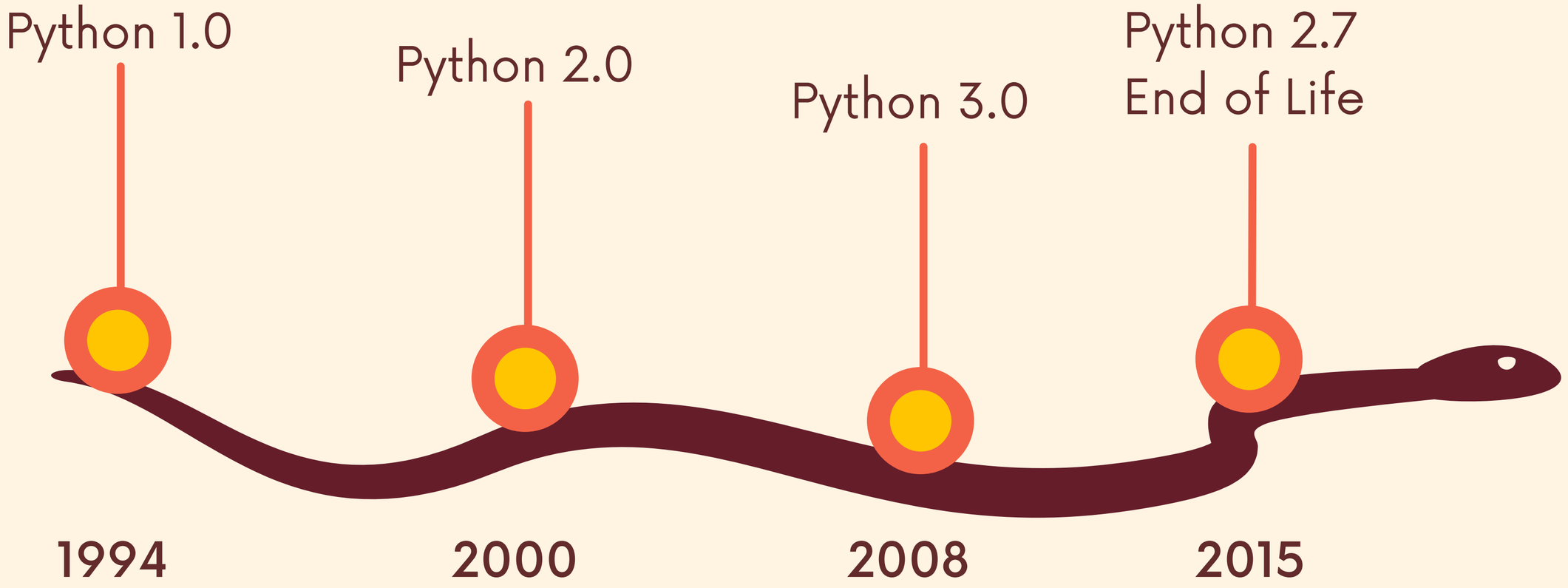
A Tiny Little Bit Of

Python History



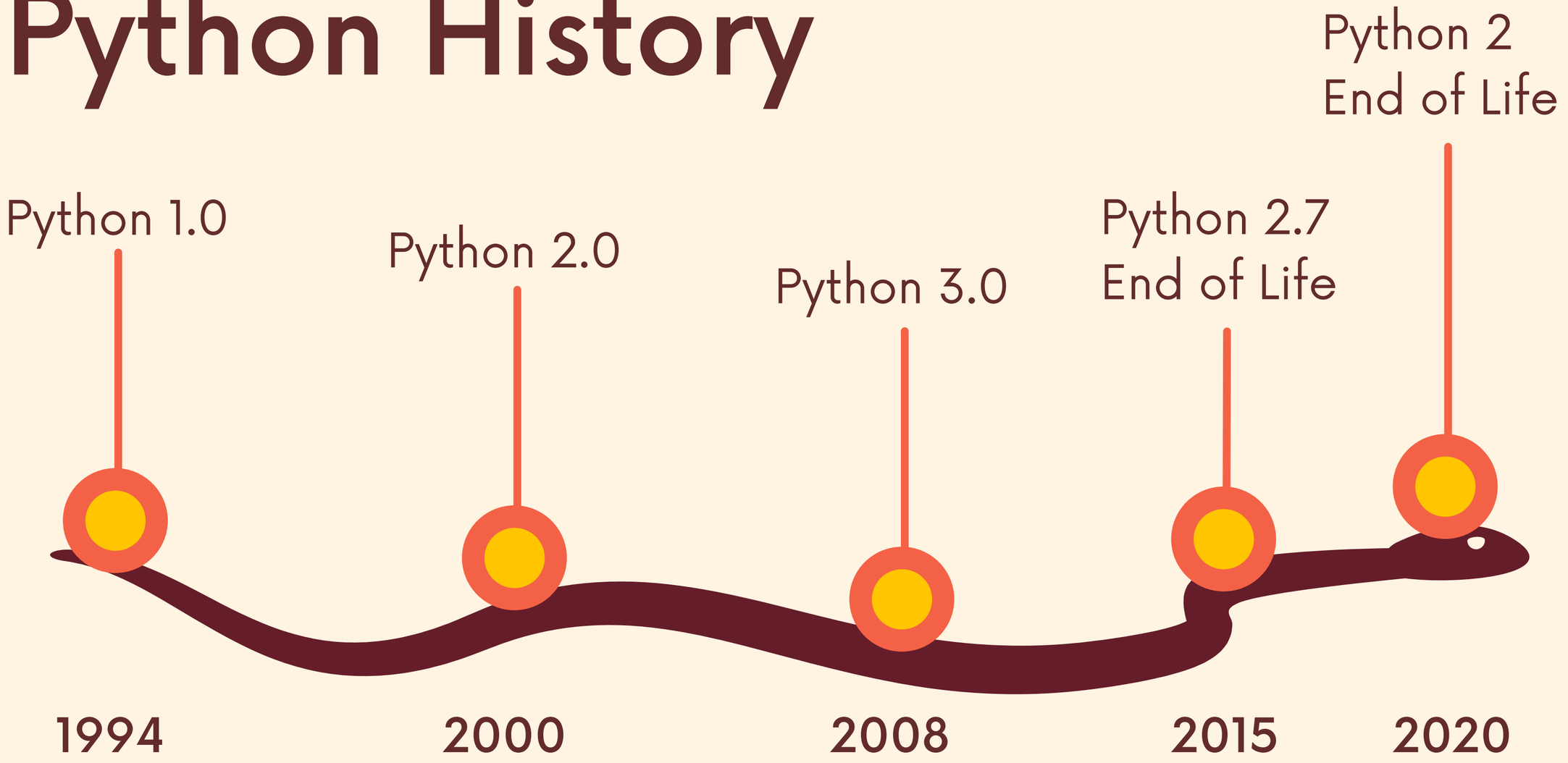
A Tiny Little Bit Of

Python History

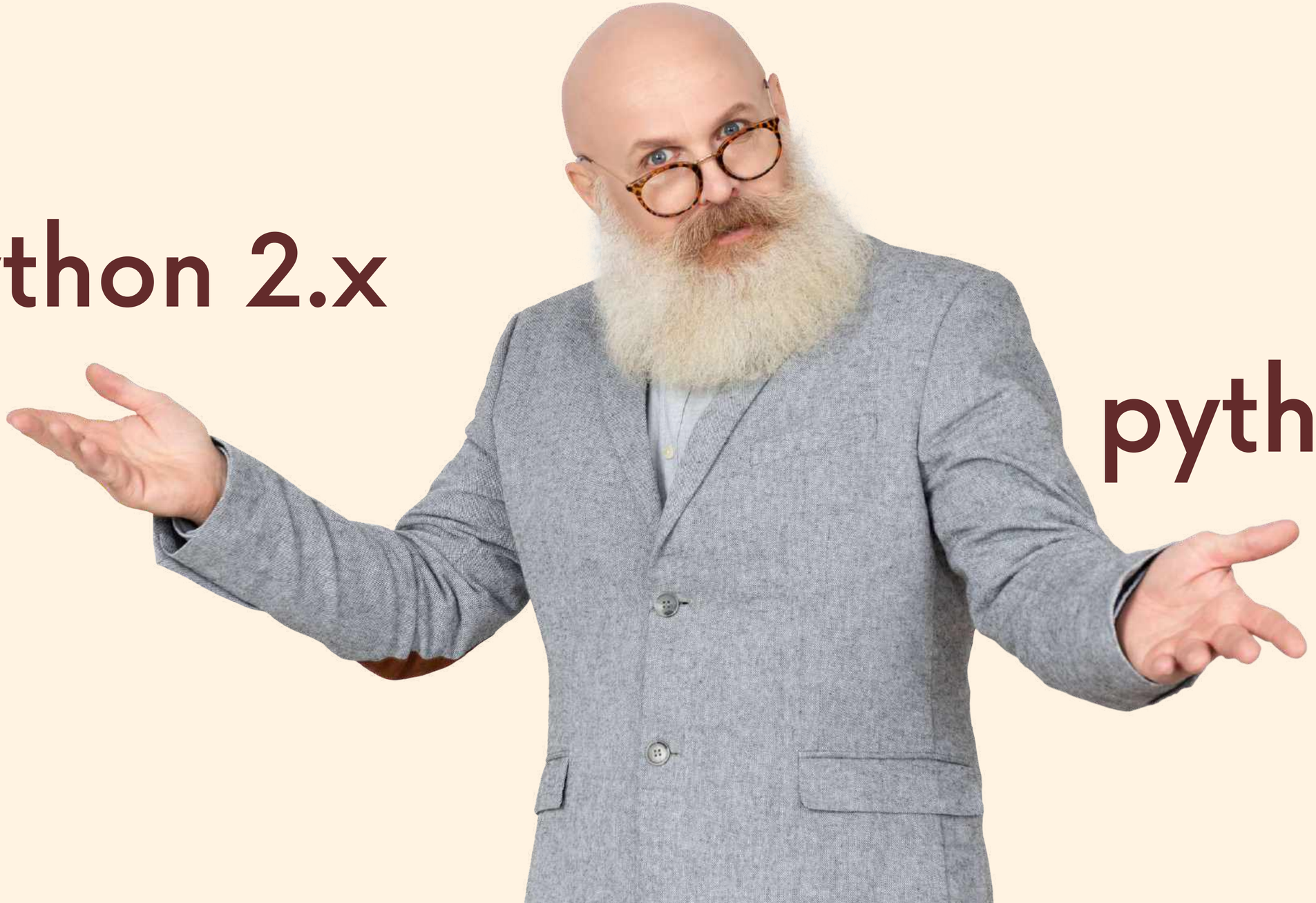


A Tiny Little Bit Of

Python History



python 2.x

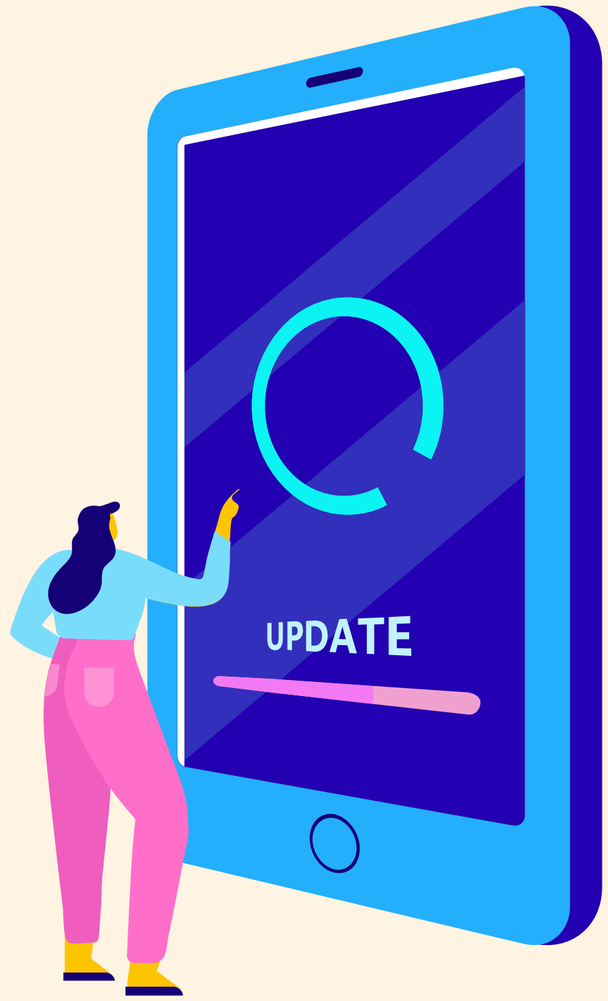


python 3.x

python 2 vs. 3

Python 3 was a major change

- **Not backwards compatible** with python 2
- It included some syntax changes
- Featured major changes under the hood
- It took 10+ years for Python 2 to be officially declared dead



python 2.x

```
>>> print "Hello"
Hello

>>> 5/2
2
```

python 3.x

```
>>> print("Hello")
Hello

>>> 5/2
2.5
```

Who Cares?

At this point, there is no real reason to learn 2.x, but it's worth knowing the history because...

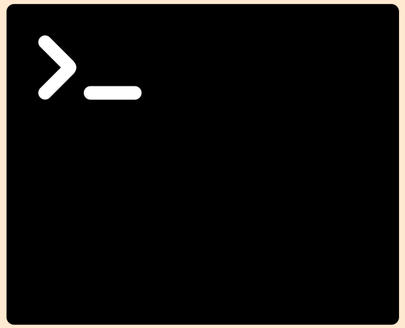
- Lots of the tutorials online are for 2.x
- Your computer may have 2.x pre-installed
- Some dead libraries/tools still haven't been updated to work with 3.x

Installation

(it's actually not bad!)

Running Python

There are two main ways we can run Python code



Interactively

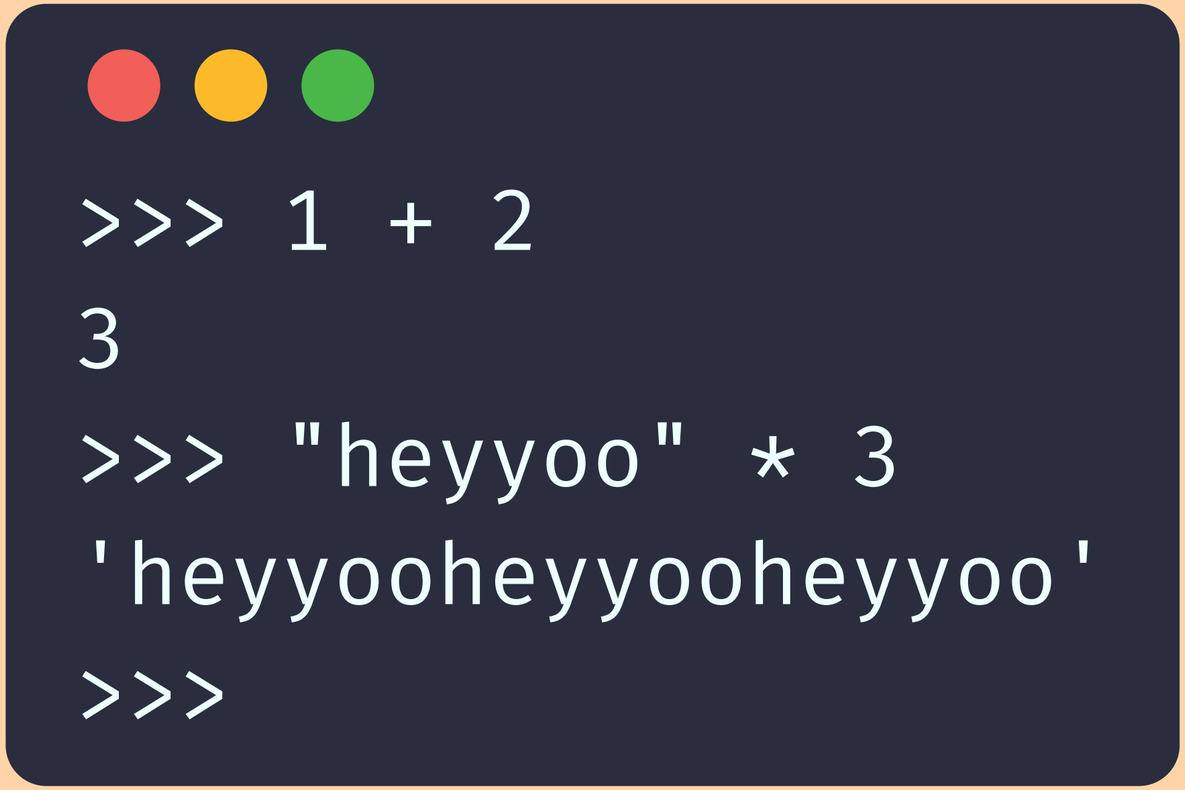
From A File

Interactive Python

In interactive mode, any code you enter is immediately run.

Python executes each line as you enter it and then displays the output.

Great for learning and trying things out, but not great for "real" applications



Python Scripts

Alternatively, you can write python code in a file.

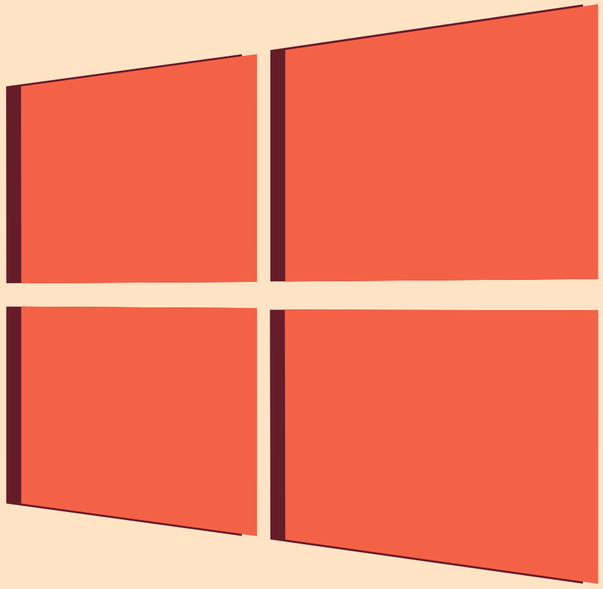
game.py

And then you can separately execute that script with Python

```
python game.py
```

The code only runs when you manually tell it to.

Mac Install



Windows Install

Visual Studio Code

The "I want to get started right now" option

repl.it

Numbers, Variables, And More

Write Your Review

Click on a star to change your rating 1 - 5, where 5 = great! and 1 = really bad

Your Review:

Your Reivew

999 Characters remaining

Your Info:

Name:

Name

Email:

Email

I Agree to the Terms blah blah blah

Submit

Basic Data Types

Strings

Integers

Booleans

Floats

Data Types

Strings	Integers	Frozenset
Booleans	Floats	Bytes
Dictionary	Complex	Range
List	Tuple	Set

Integers

- Whole numbers only
- Positive or Negative
- No Decimal Points!

Integers

- Whole numbers only
- Positive or Negative
- No Decimal Points!

9

Integers

- Whole numbers only
- Positive or Negative
- No Decimal Points!

9

378

Integers

- Whole numbers only
- Positive or Negative
- No Decimal Points!

9	378	-21

Integers

- Whole numbers only
- Positive or Negative
- No Decimal Points!

9	378	-21

Floats

- Written With a Decimal Point
- Positive or Negative

Integers

- Whole numbers only
- Positive or Negative
- No Decimal Points!

9

378

-21

Floats

- Written With a Decimal Point
- Positive or Negative

1.5

Integers

- Whole numbers only
- Positive or Negative
- No Decimal Points!

9

378

-21

Floats

- Written With a Decimal Point
- Positive or Negative

1.5

9.99

Integers

- Whole numbers only
- Positive or Negative
- No Decimal Points!

9	378	-21

Floats

- Written With a Decimal Point
- Positive or Negative

1.5	9.99	-2.0

INT

1234987234321

+8

0

378

1_000_000_000

-99

-8363247238498

1782

FLOAT

1.3333333

0.5

+8.1

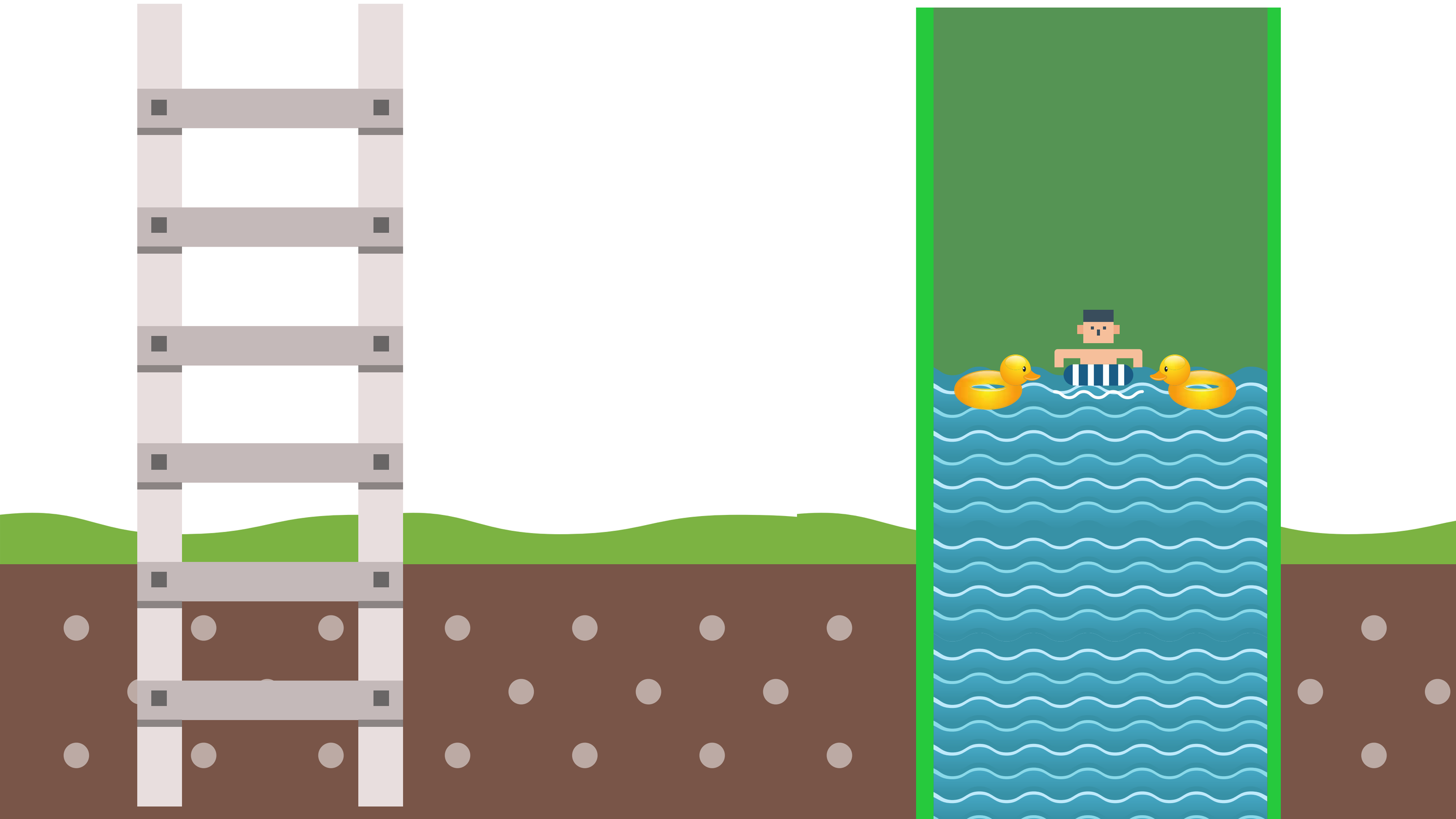
-.2

0.0

0.8372984732894733

1.234987234321e12

04.0



Ints

1_000

777

1,000

0777

Floats

1_000.55

1.234e12

1,000.55

1.234 e12

Operators

Operators are special characters in Python that perform operations on value(s). Below are some of the most common:

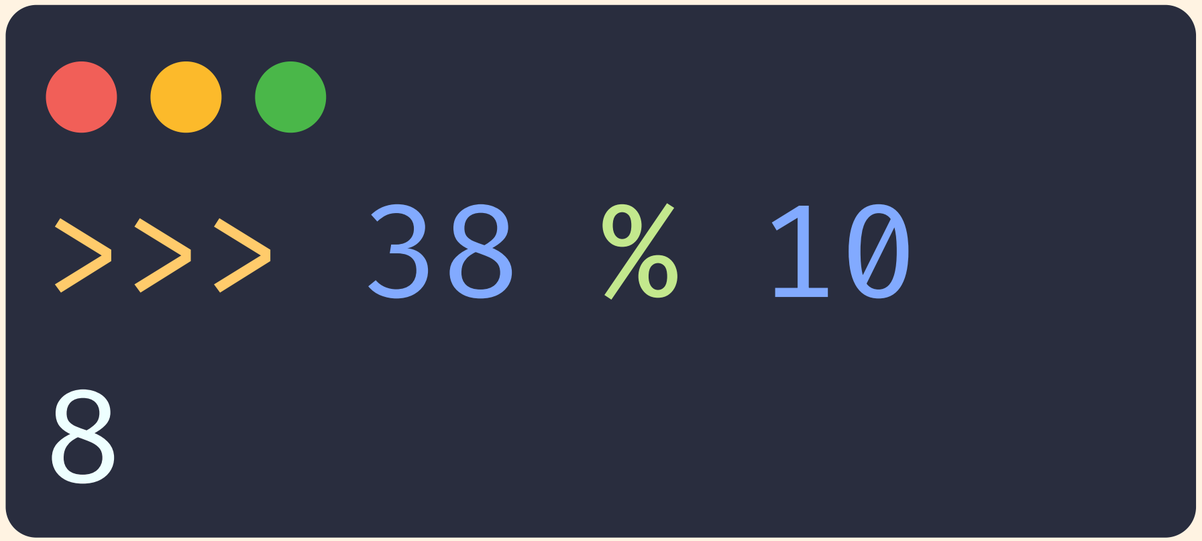
+	*	>	<=	and	is	=	*=
-	/	<	==	or	in	+=	/=
**	%	>=	!=	not	!=	-=	|=

Order of Operations

Parentheses	( )
Multiplication and Division	* / //
Addition and Subtraction	+ -

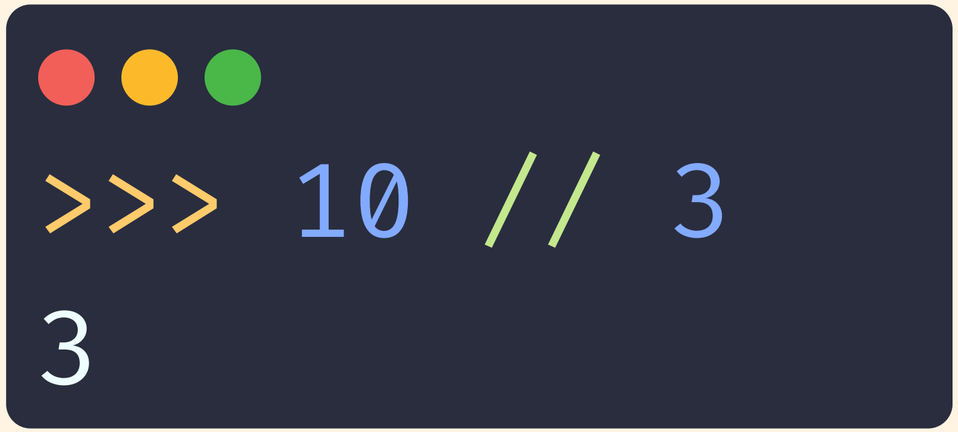
Integer Division	//
Exponentiation	**
Modulo	%

Modulo



10 goes into 38...
3 times, with a remainder of 8

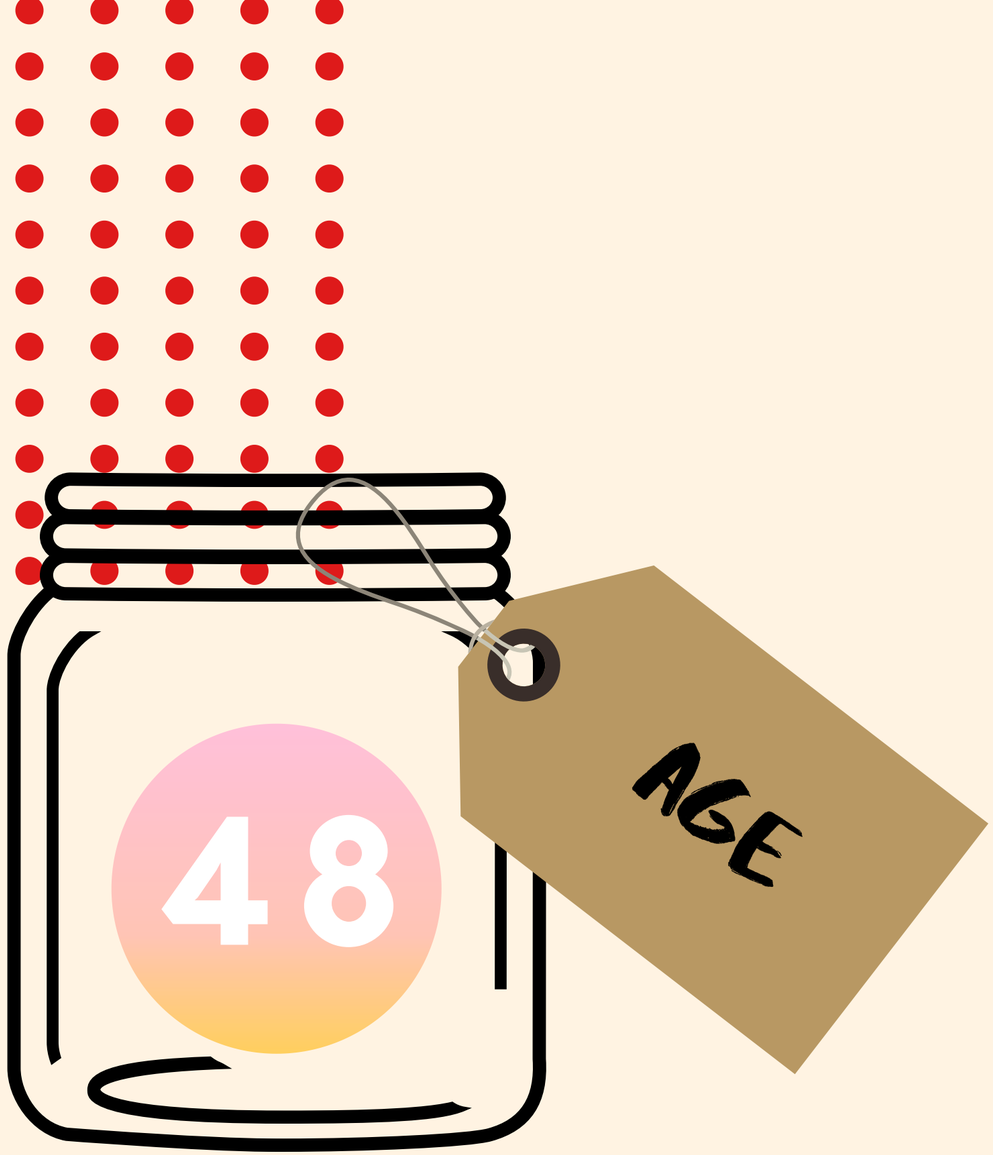
Integer Division



Comments

```
# this never runs
```

Python will ignore any lines starting with the # symbol



Variables

VARIABLES ARE LIKE LABELS FOR VALUES

We can store a value and give it a name so that we can:

- Refer back to it later
- Use that value to do...stuff
- Change it later on

Potential Variables

num_guesses → 0

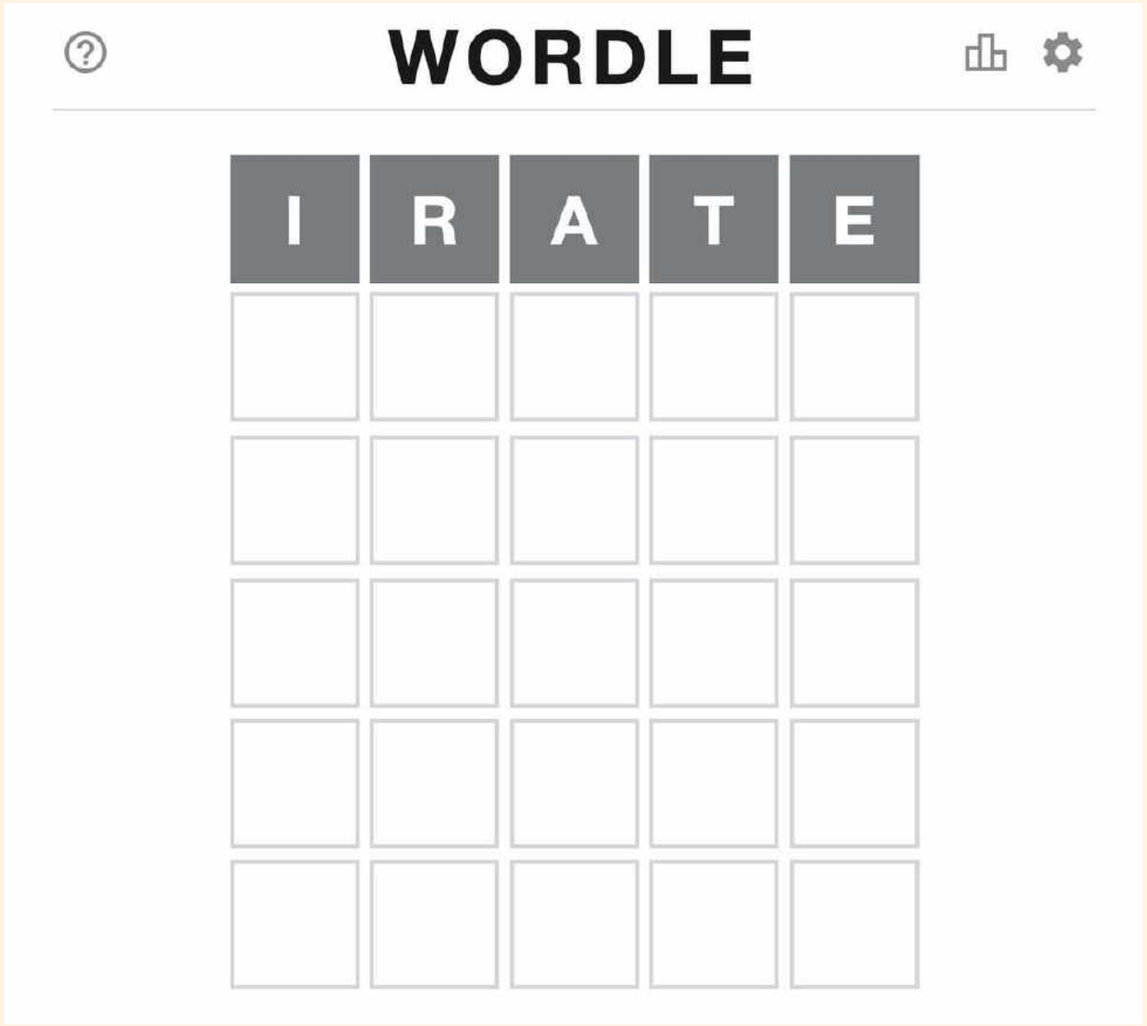
target_word → "'knoll"

max_guesses → 6

guesses →

correct_letters →

game_over → False



Potential Variables

num_guesses → 1

target_word → "'knoll"

max_guesses → 6

guesses → ""irate"

correct_letters →

game_over → False

Potential Variables

num_guesses → 2

target_word → "'knoll"

max_guesses → 6

guesses → ""irate", "sound"

correct_letters → "o", "n"

game_over → False

Potential Variables

num_guesses → 3

target_word → "knoll"

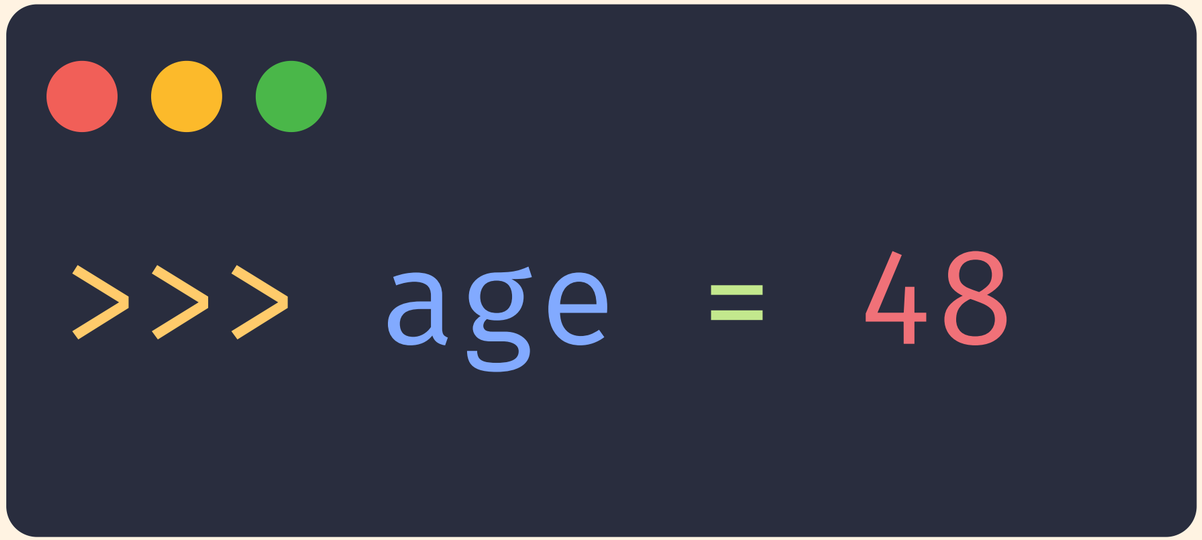
max_guesses → 6

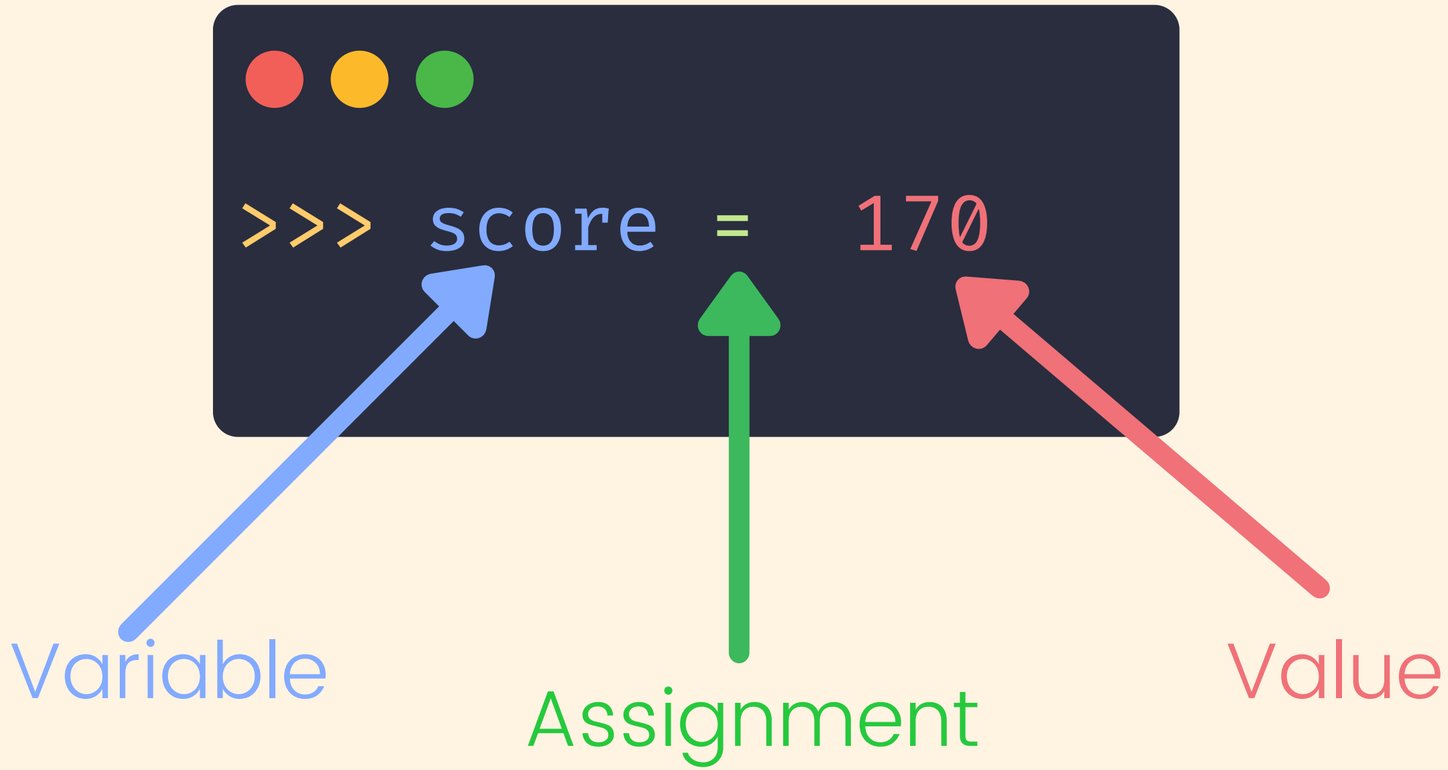
guesses → ""irate", "sound", "knoll"

correct_letters → "o", "n", "k", "l"

game_over → True

Variables




>>> score = 170
Variable
Assignment
Value


Python created
memory location
4303380936
HELLO
my name is
score
170

variable123	123variable	l
first_name	first name	O
player_1	False	def
		x
		FirstName
		FIRSTNAME

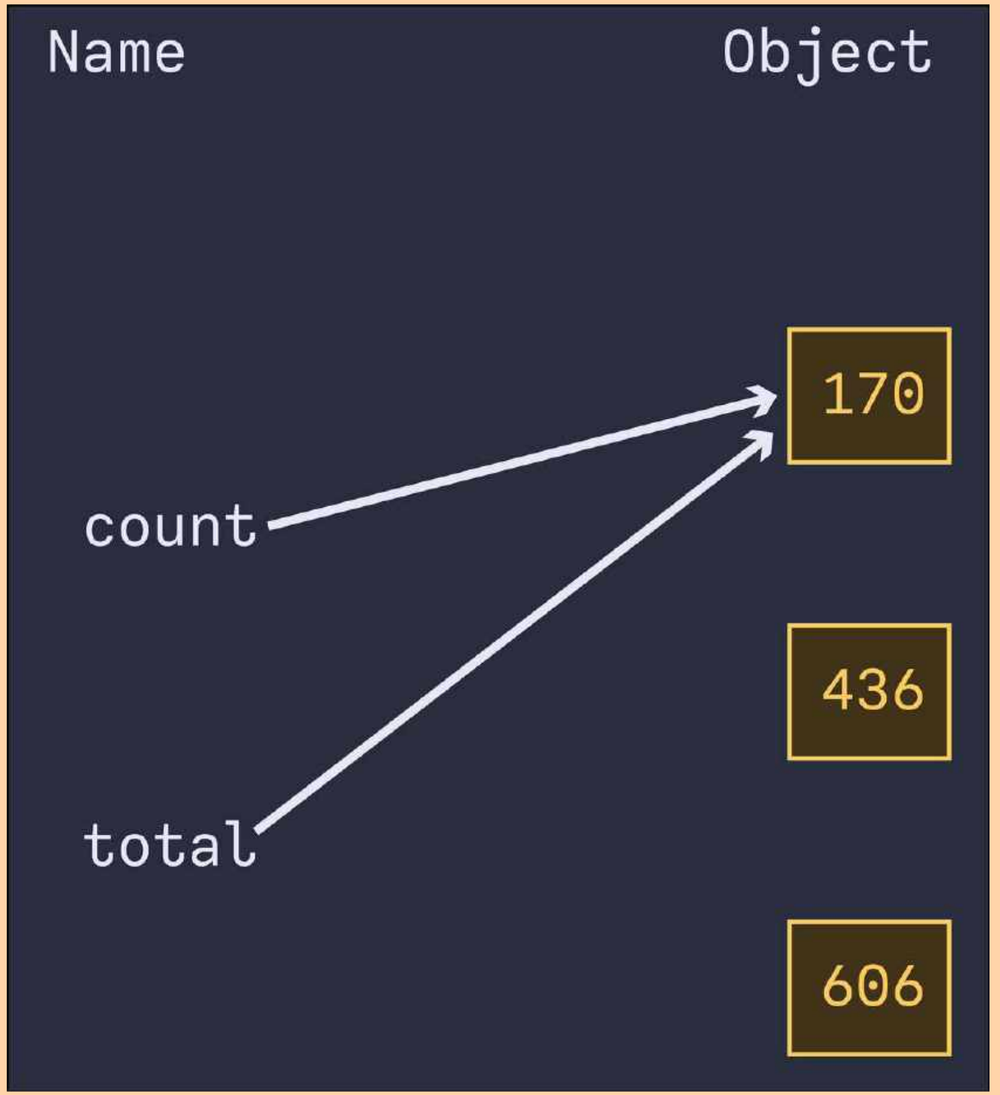
Python Keywords

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

```
False   raise   in          not       global
True    try     is          with      del
None    except  as          or        assert
for     import  and         class
while   from    continue    def
if      pass    yield       finally
else    break   await       lambda
elif    return  async       nonlocal
```

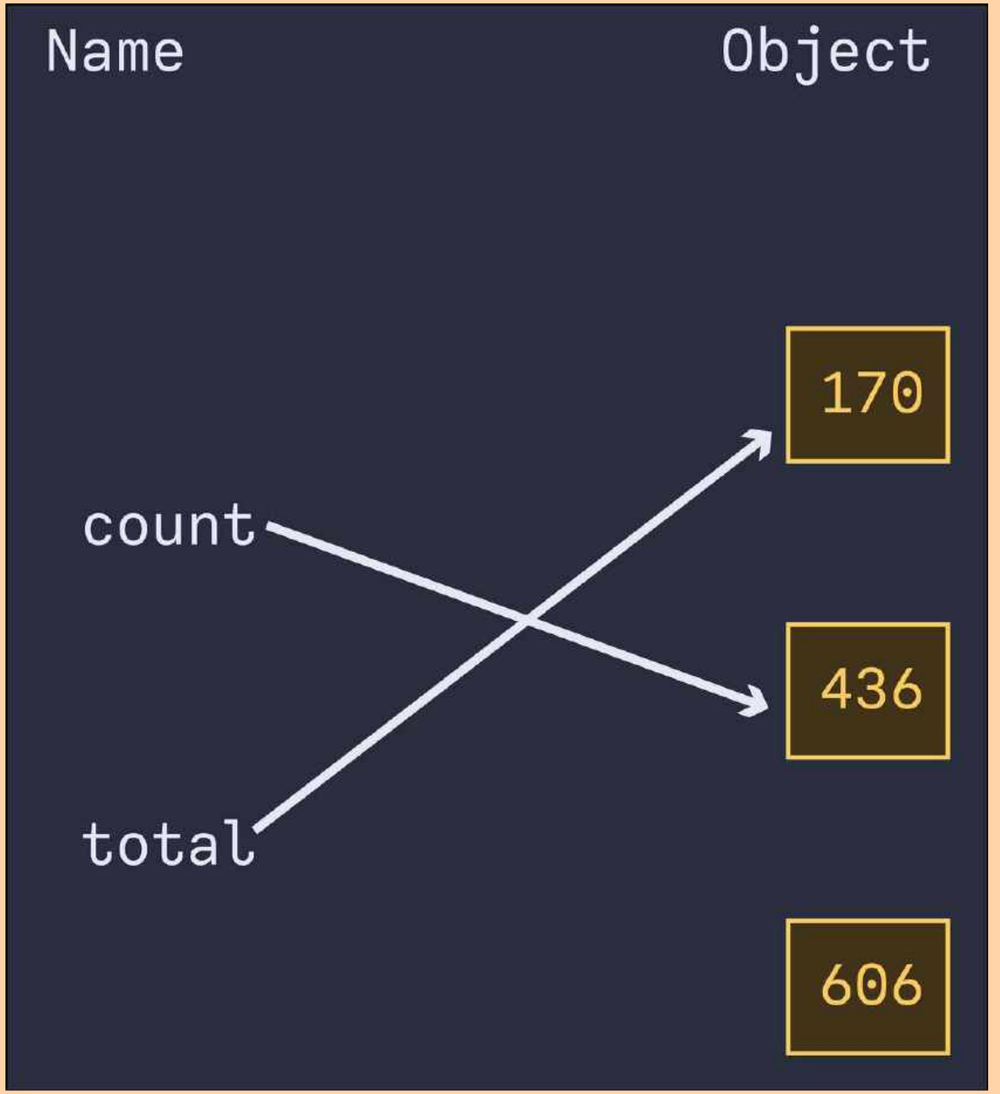
How Variables Work

```
>>> count = 170

>>> total = 170
```



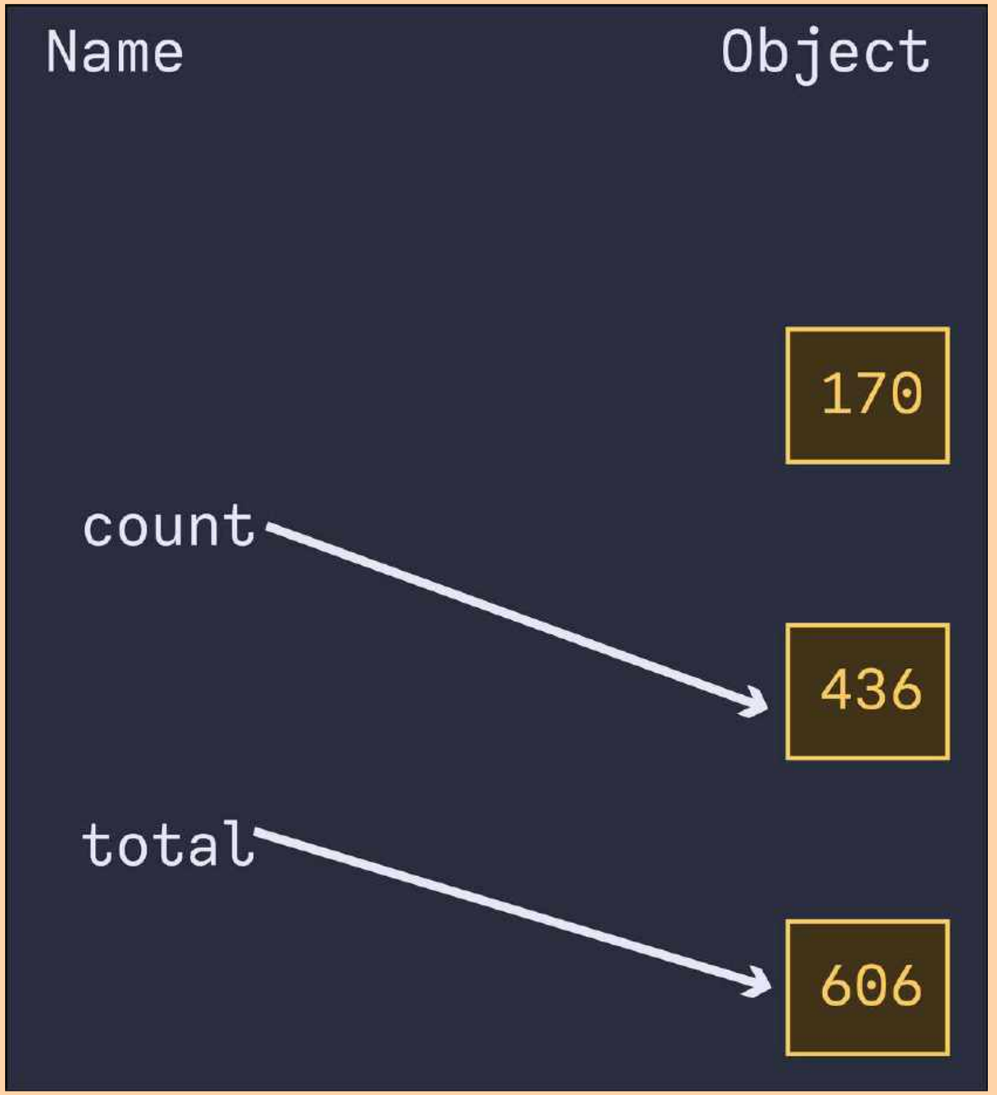
How Variables Work

```
>>> count = 170

>>> total = 170

>>> count = 436
```



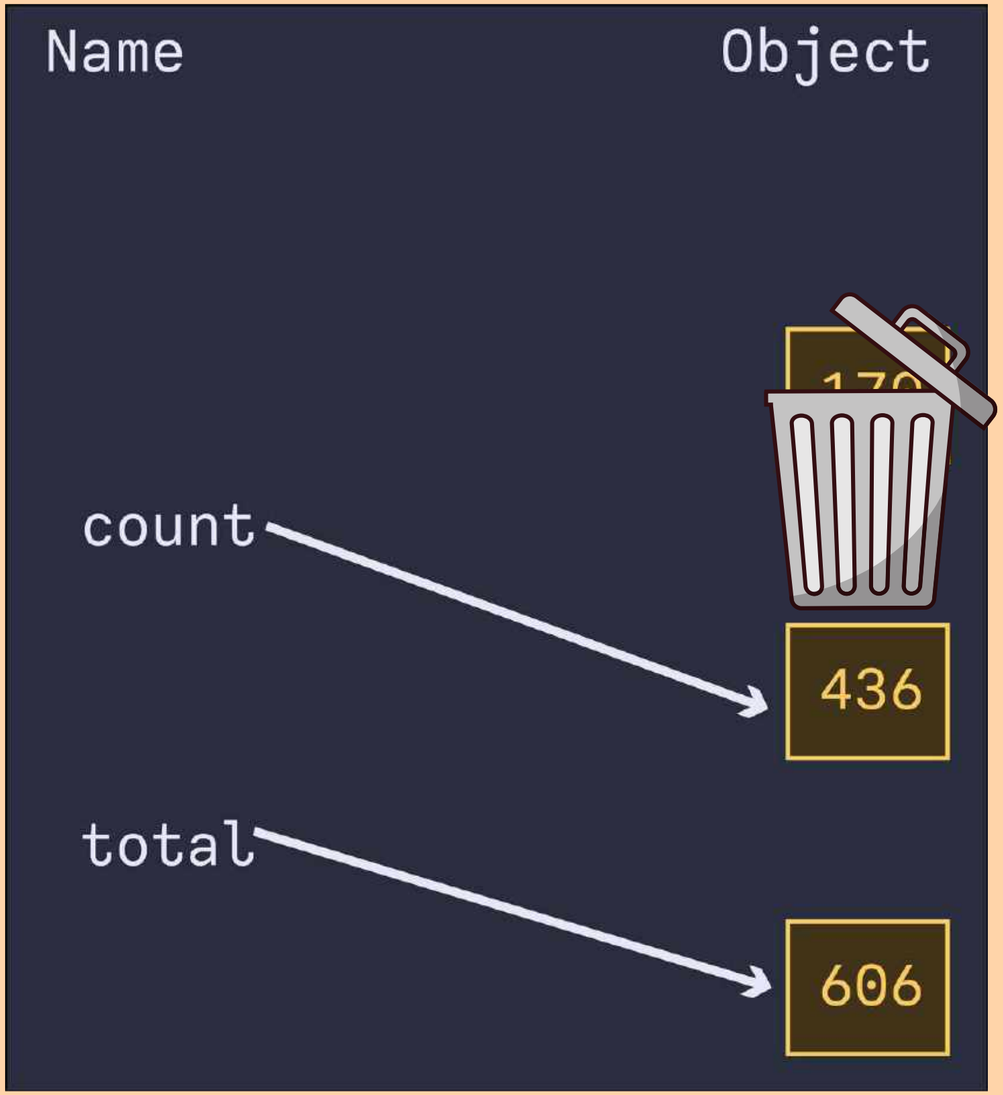
How Variables Work

```
>>> count = 170
>>> total = 170
>>> count = 436
>>> total = 606
```



How Variables Work

```
>>> count = 170
>>> total = 170
>>> count = 436
>>> total = 606
```



Assignment Operators

+=

-=

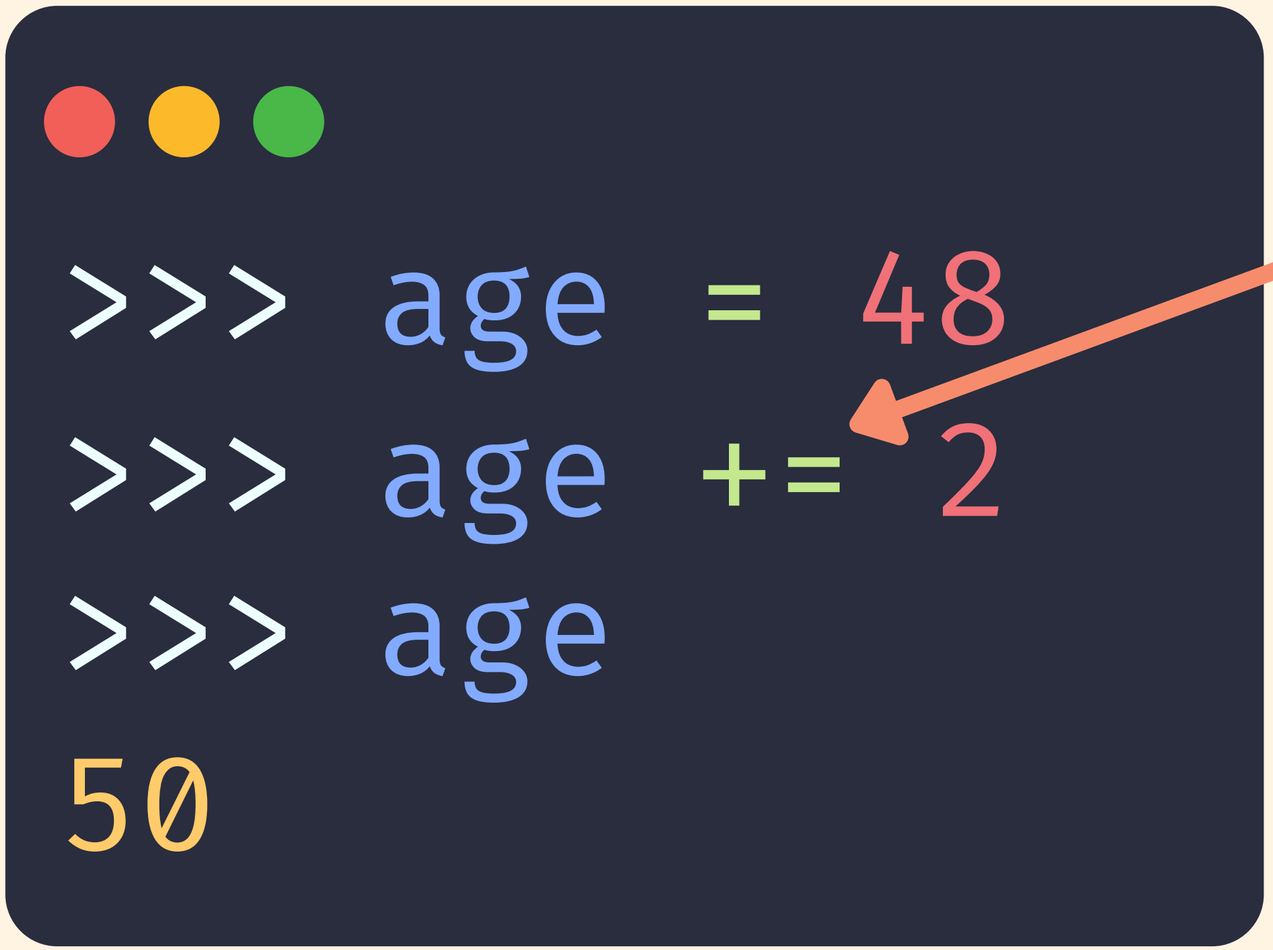
*=

/=

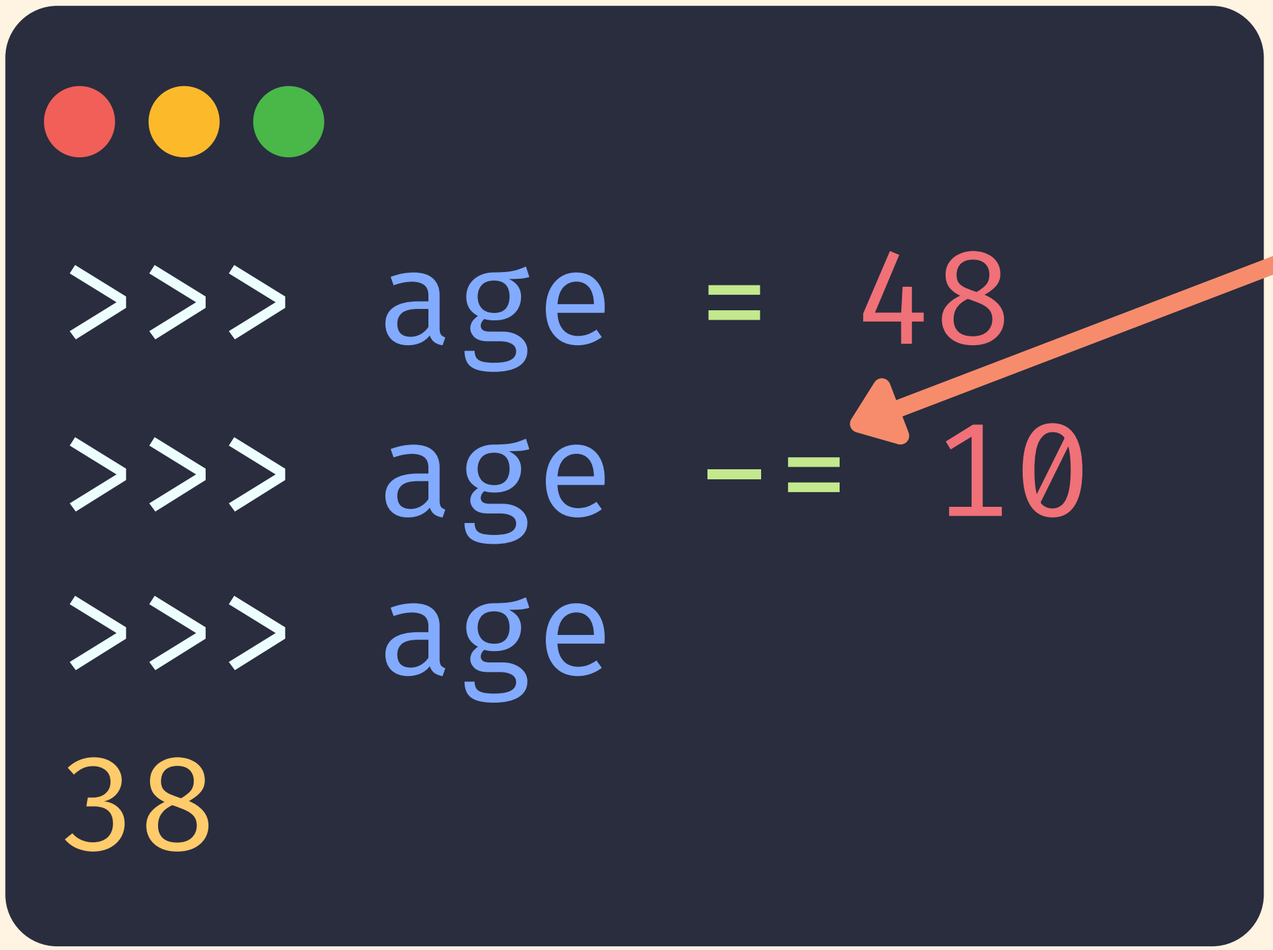
//=

**=

%=



Update age to be its current value (48) plus 2



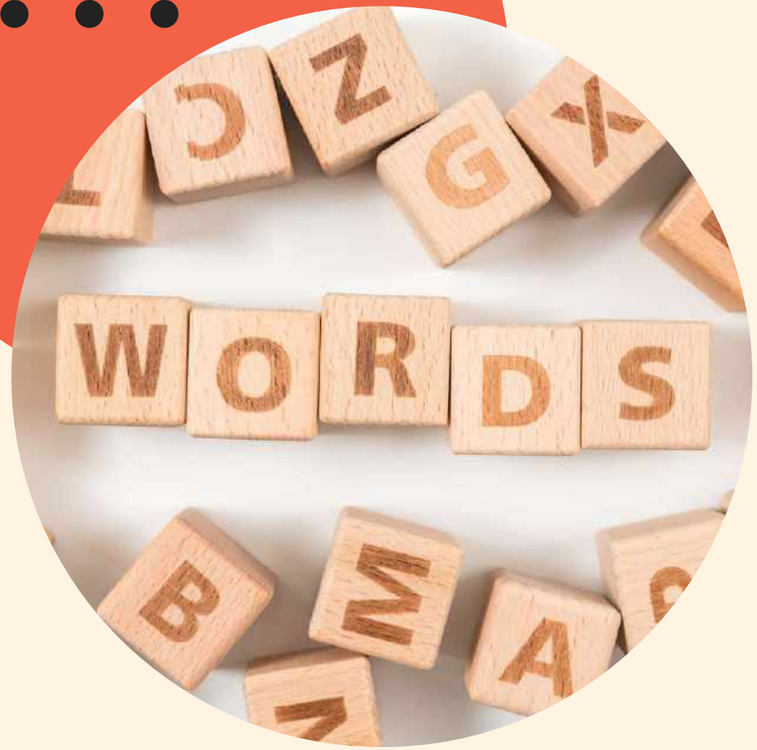
Update age to be its current value (48) minus 10

Strings

Strings

Basic Data Types

- Strings
- Integers
- Booleans
- Floats

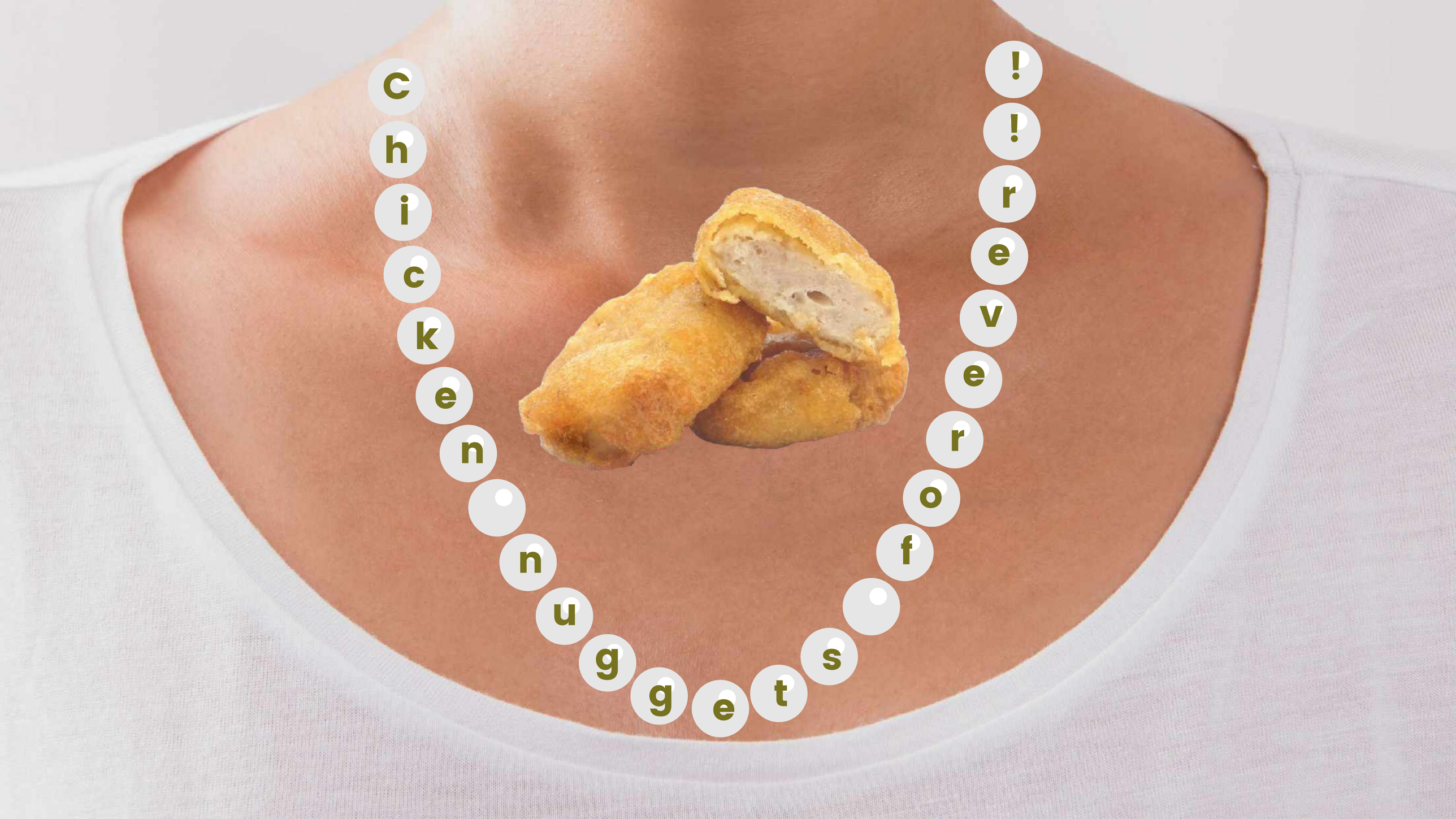


Strings

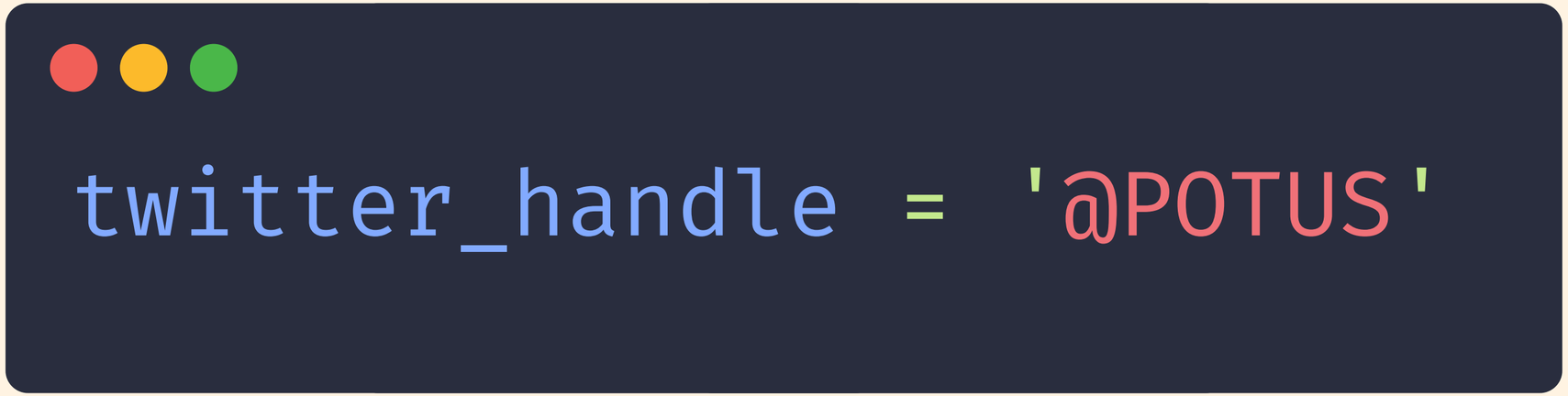
"STRINGS OF CHARACTERS"

Strings are a textual datatype and must be wrapped in quotes


chicken nugget sforever!!

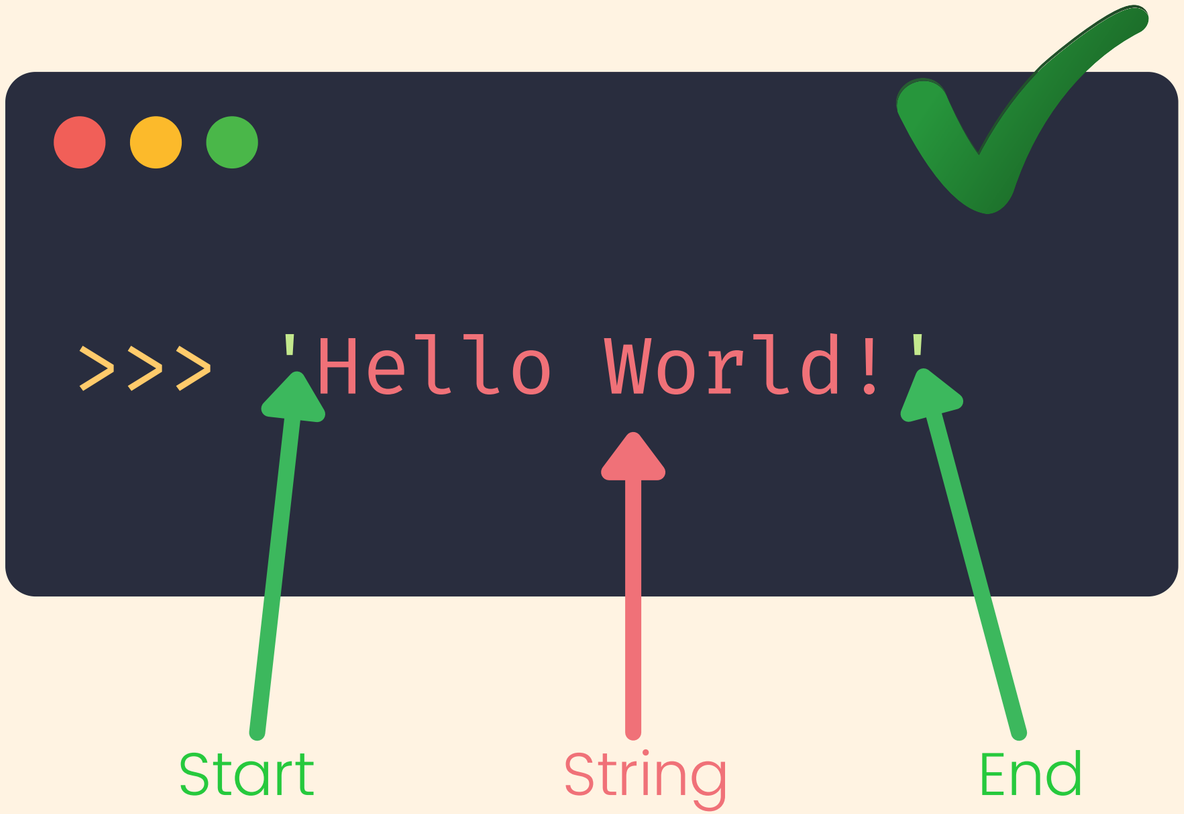

chicken nugget forever!!

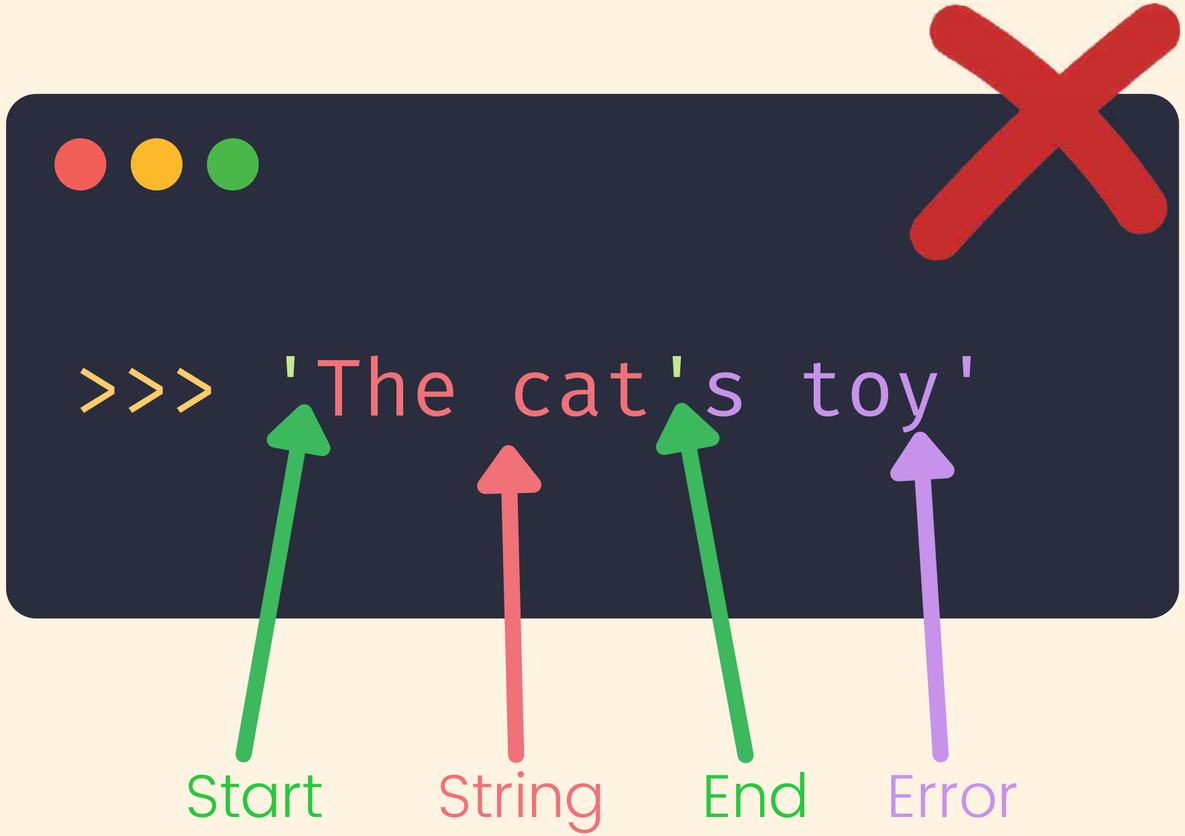

color = "Magenta"


twitter_handle = '@POTUS'

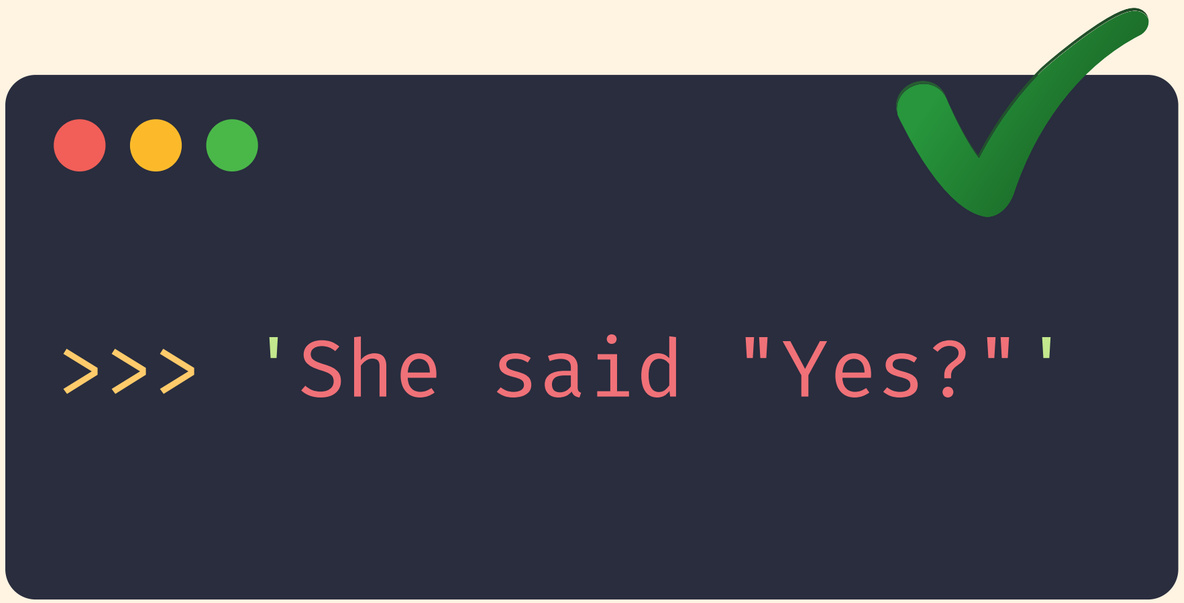

url = "www.reddit.com/r/formula1/"

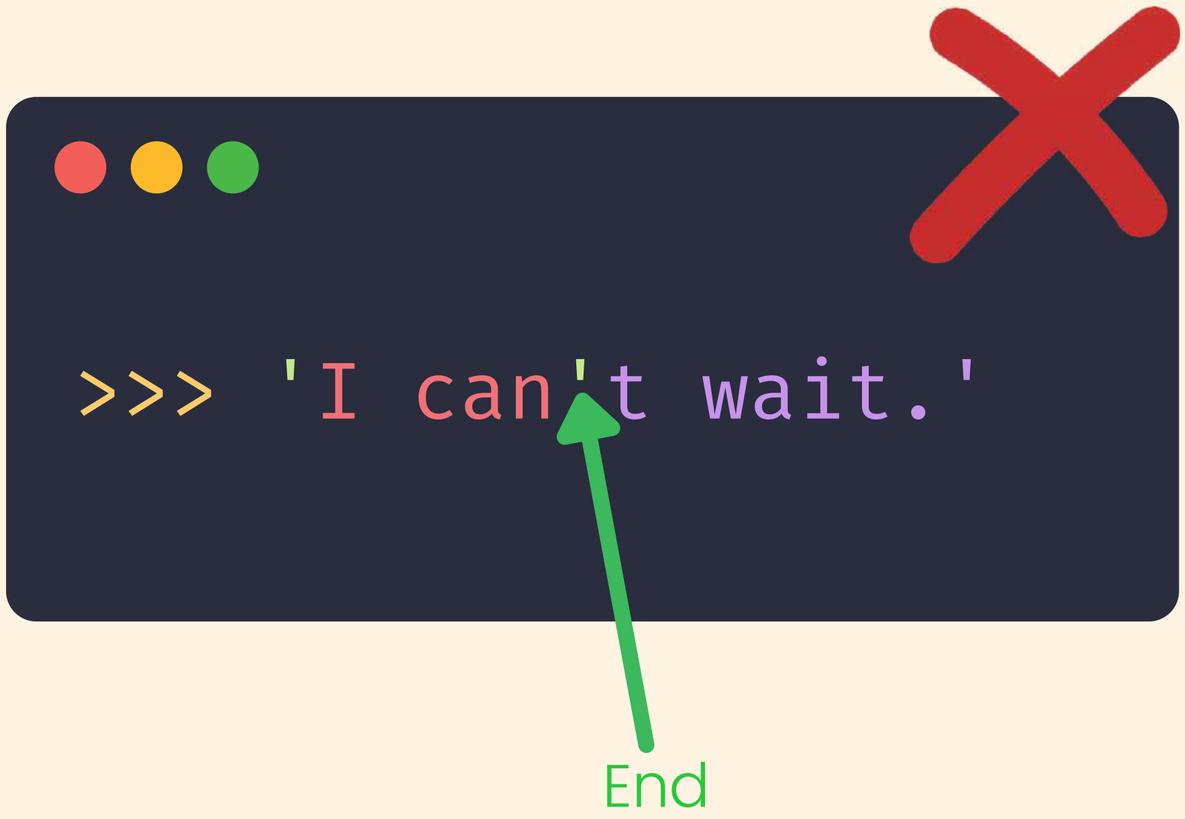
Quotes


>>> 'Hello World!'
Start
String
End

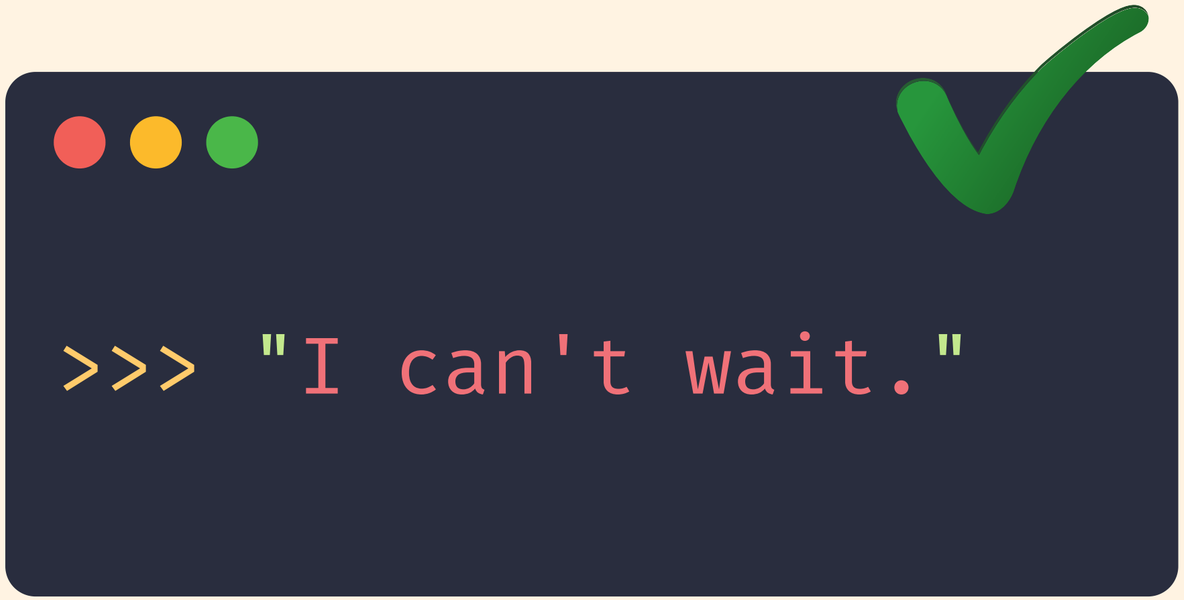

>>> 'The cat's toy'
Start
String
End
Error

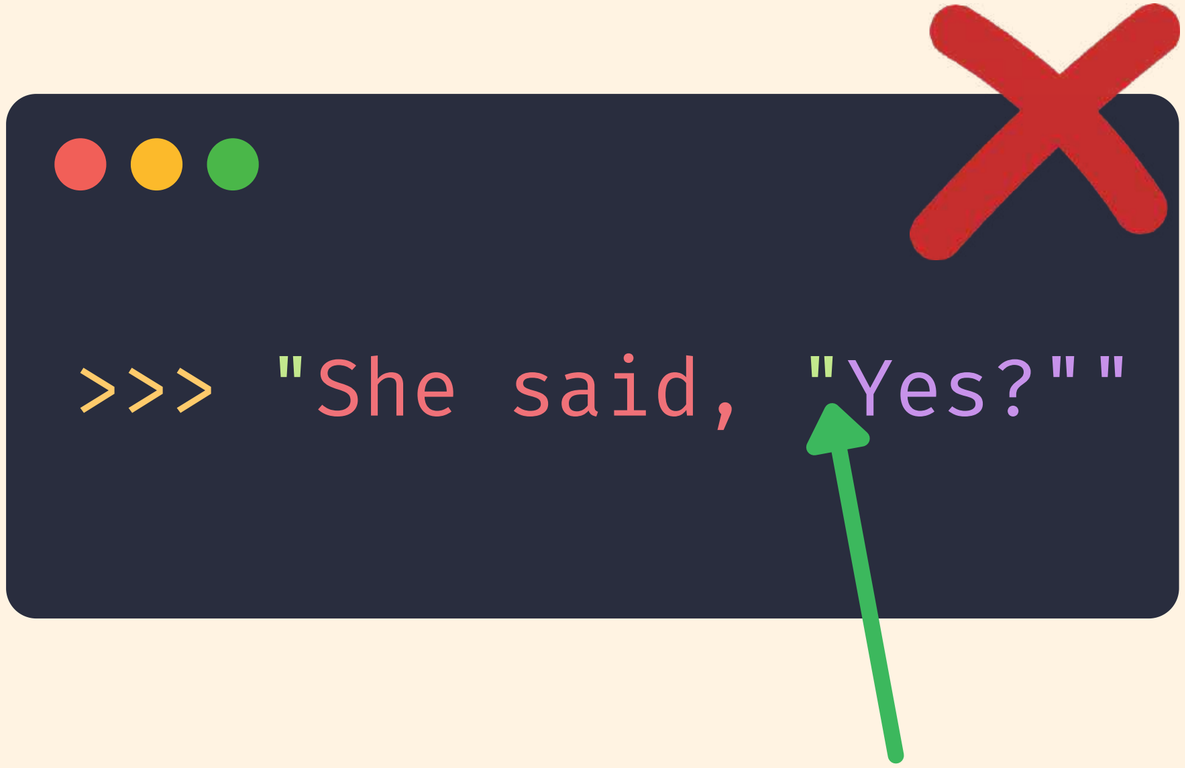
Single Quotes





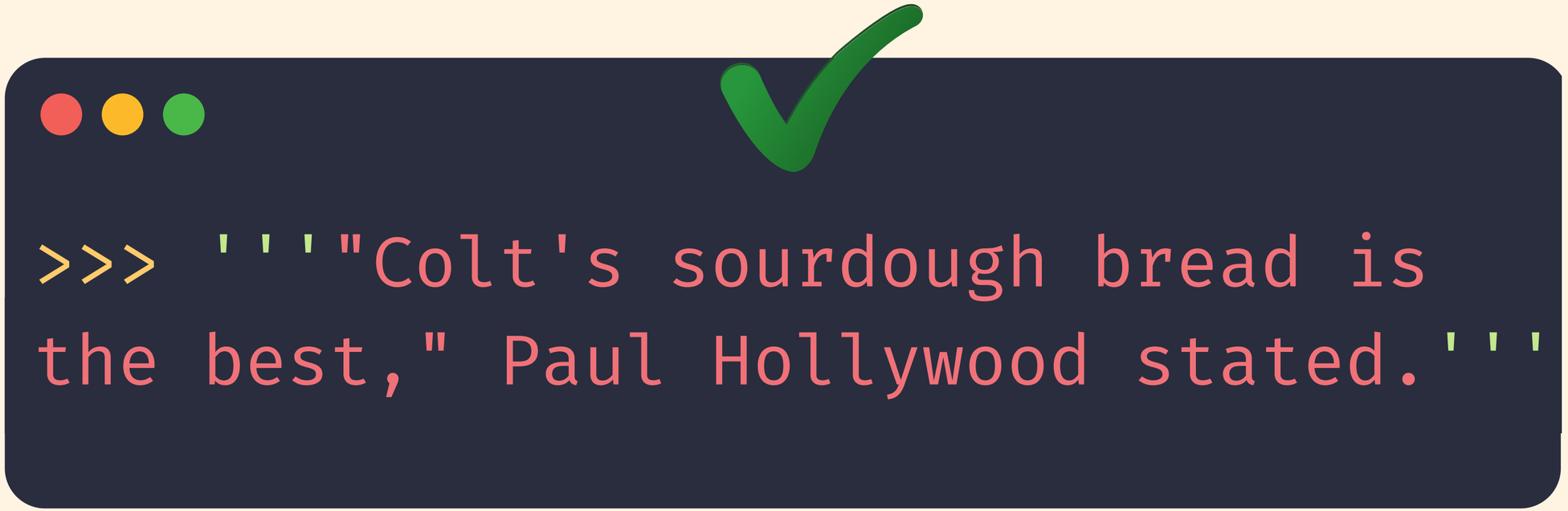
Double Quotes





Triple Quotes

Single Or Double



```
>>> '''"Colt's sourdough bread is
the best," Paul Hollywood stated.'''
```

Print

The **print()** function prints out any values we pass to it to "standard output". It does not return anything.

Escape Characters

Newline - \n

Double Quote - \"

Single Quote - \'

Tab - \t

Backslash - \\

Concatenation

We can concatenate strings together by using the plus sign. No space will be added between them.

```
>>> 'pan' + 'cake'
'pancake'
```

Multiplication

We can also multiply a string by a number,
which will repeat that string.

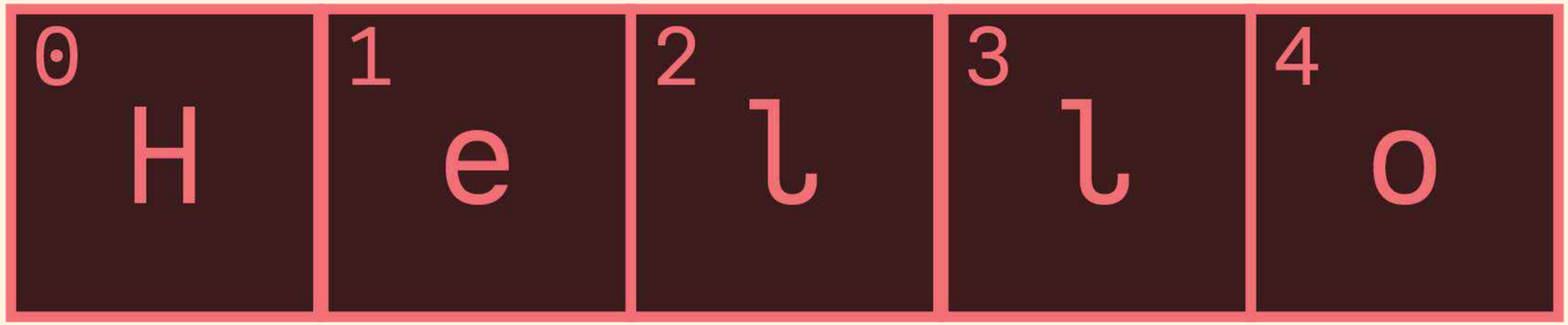
```
>>> 'ha'* 4
'hahahaha'
```

None

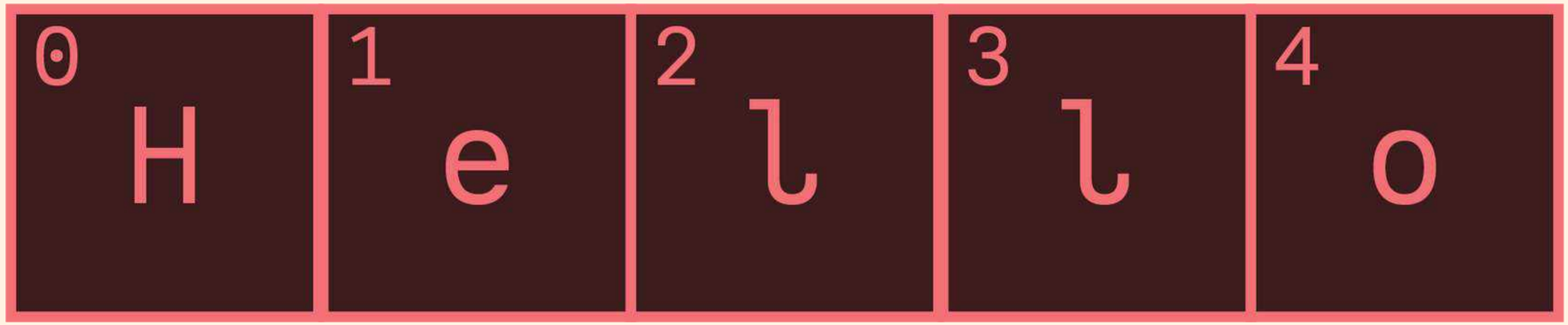
None is a special value in Python that denotes the **lack of value.** It is not the same as zero or an empty string (those are still values).

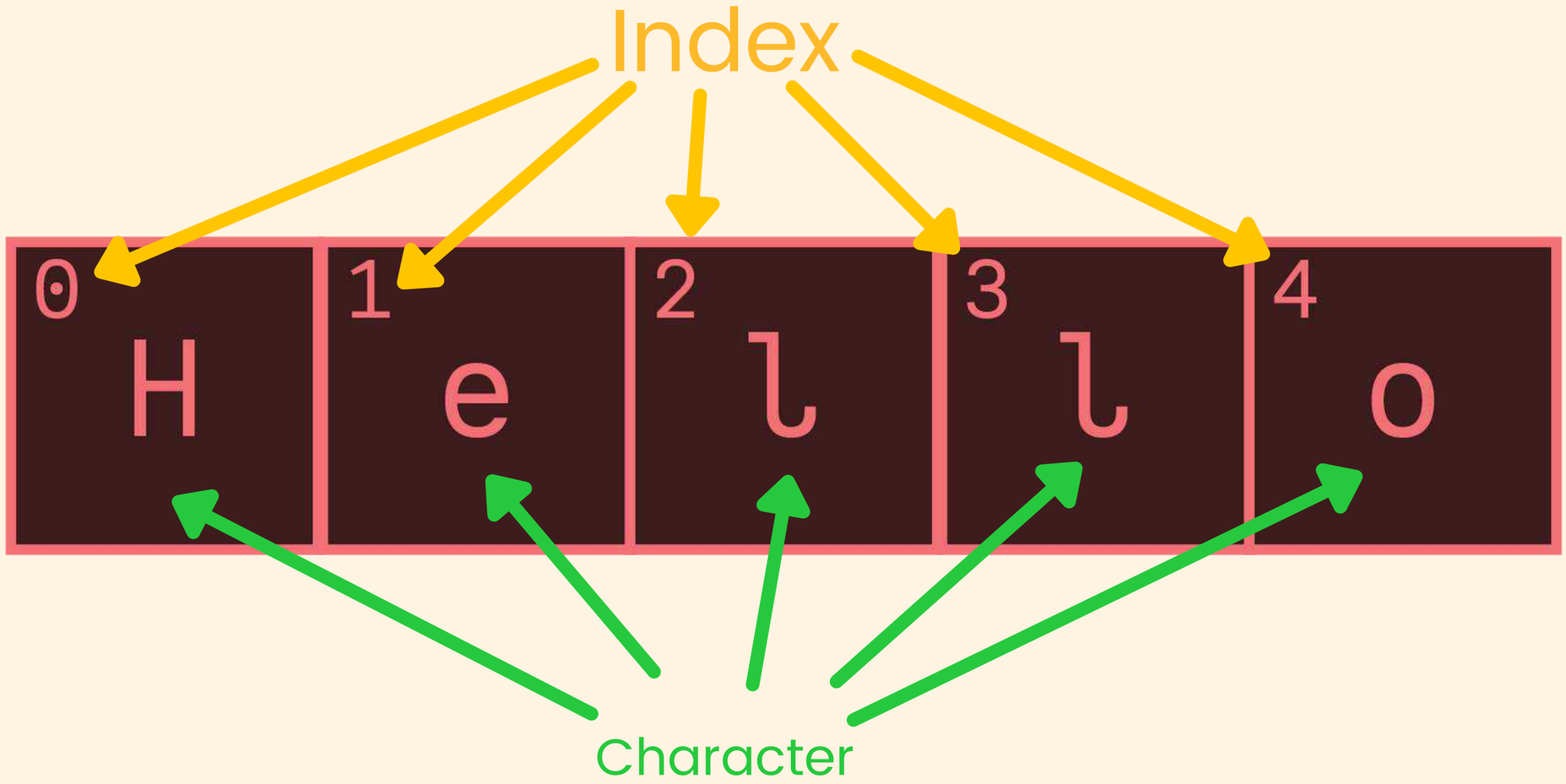
```
>>> user = None
```

Strings Are Ordered



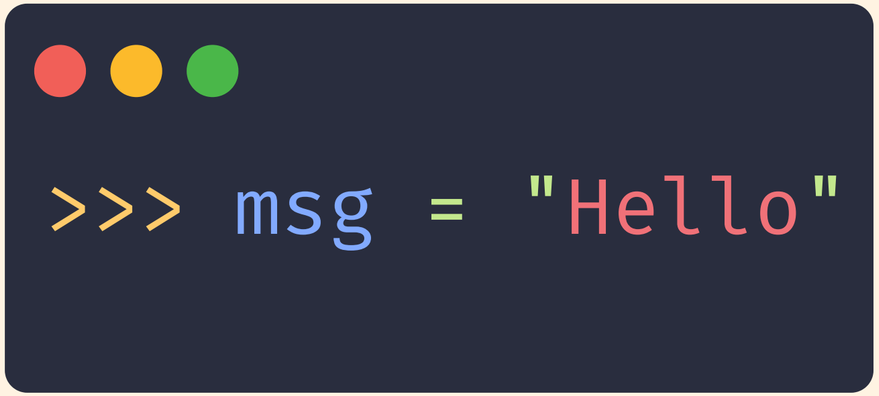
Strings Are **Indexed**




Index
0
H
1
e
2
l
3
l
4
o
Character

What is a name?

What is a name?
That which we
call a string
by any other name
would still say the same thing.



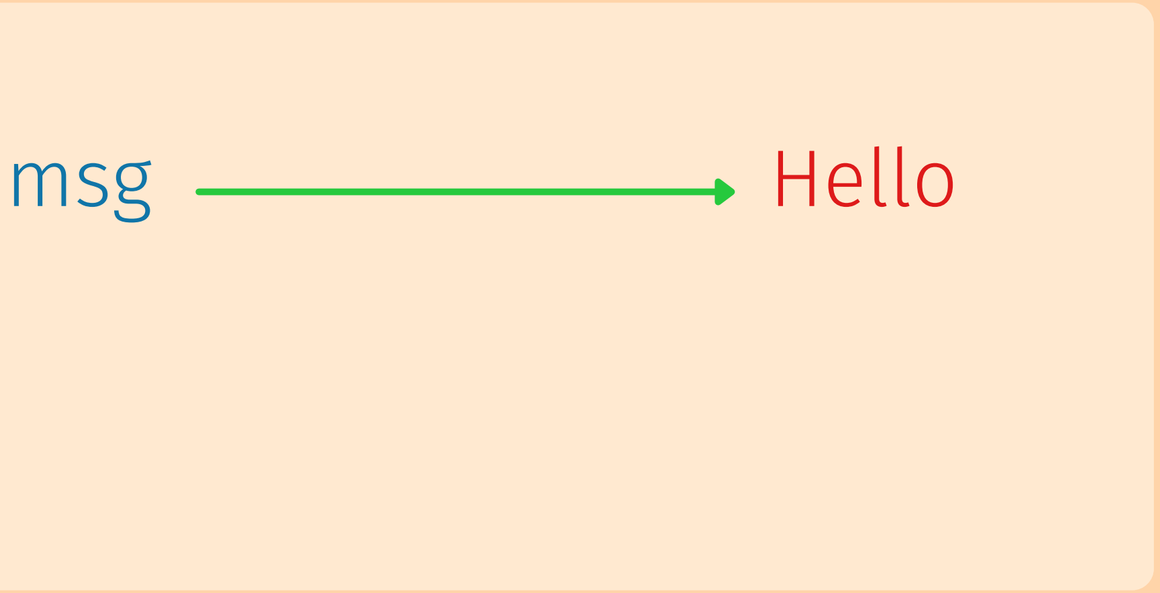
How Variables Work

Your Code

```
>>> msg = "Hello"
```

Names

Objects



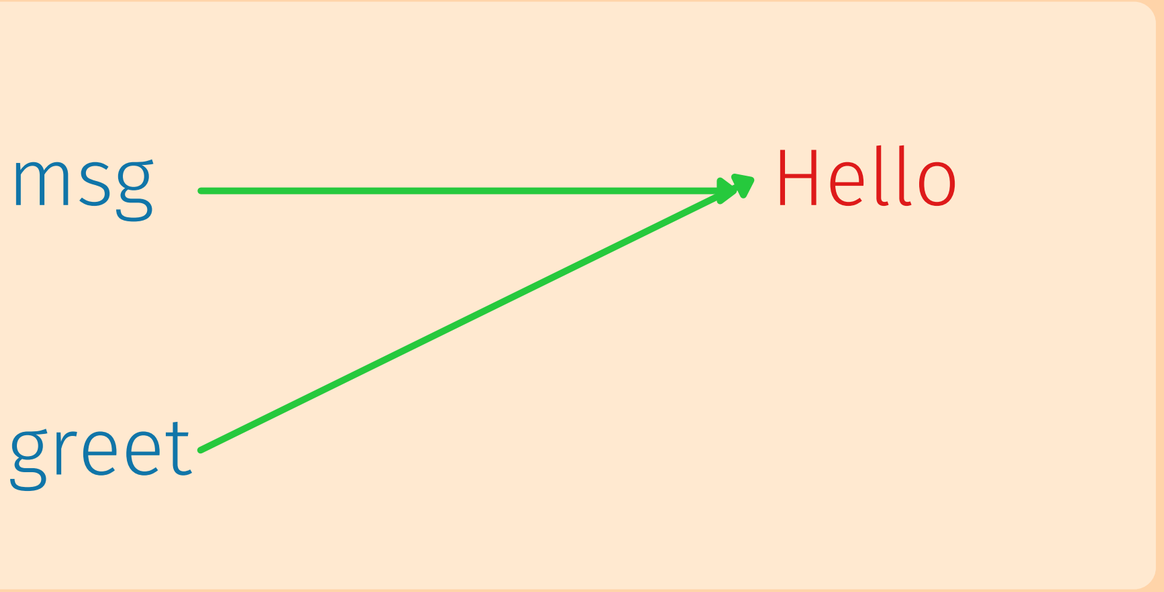
How Variables Work

Your Code

```
>>> msg = "Hello"
>>> greet = "Hello"
```

Names

Objects



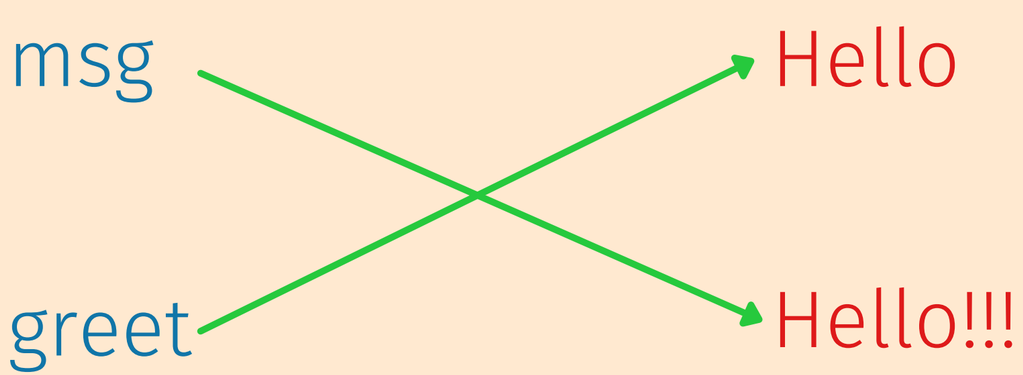
How Variables Work

Your Code

```
>>> msg = "Hello"
>>> greet = "Hello"
>>> msg = "Hello!!!"
```

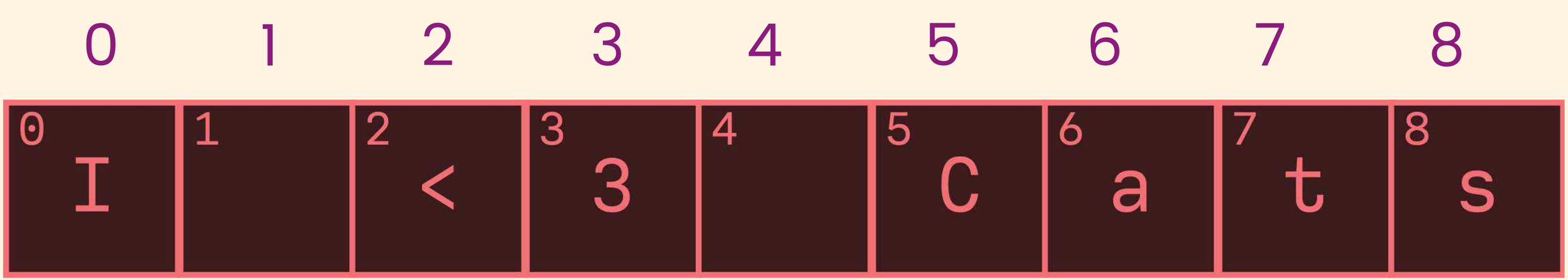
Names

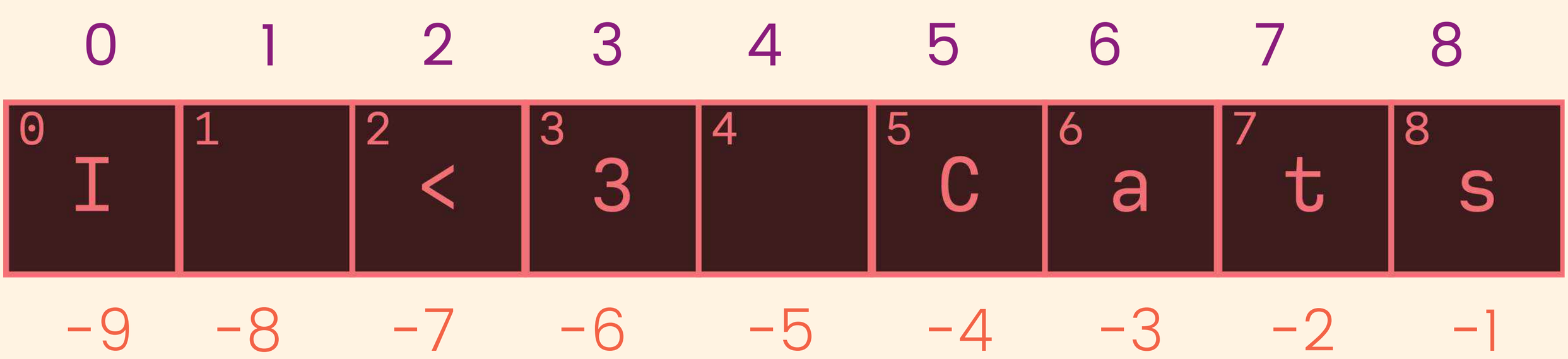
Objects



Indexes

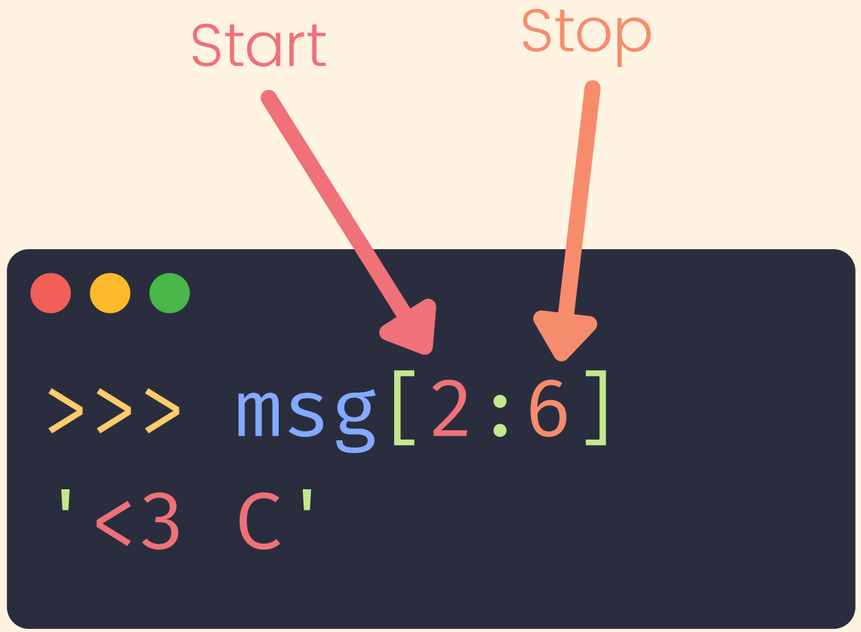

```
>>> msg = "I <3 Cats"
>>> msg[0]
'I'
>>> msg[5]
'C'
```


0 1 2 3 4 5 6 7 8
0 I
1
2 <
3 3
4
5 C
6 a
7 t
8 s

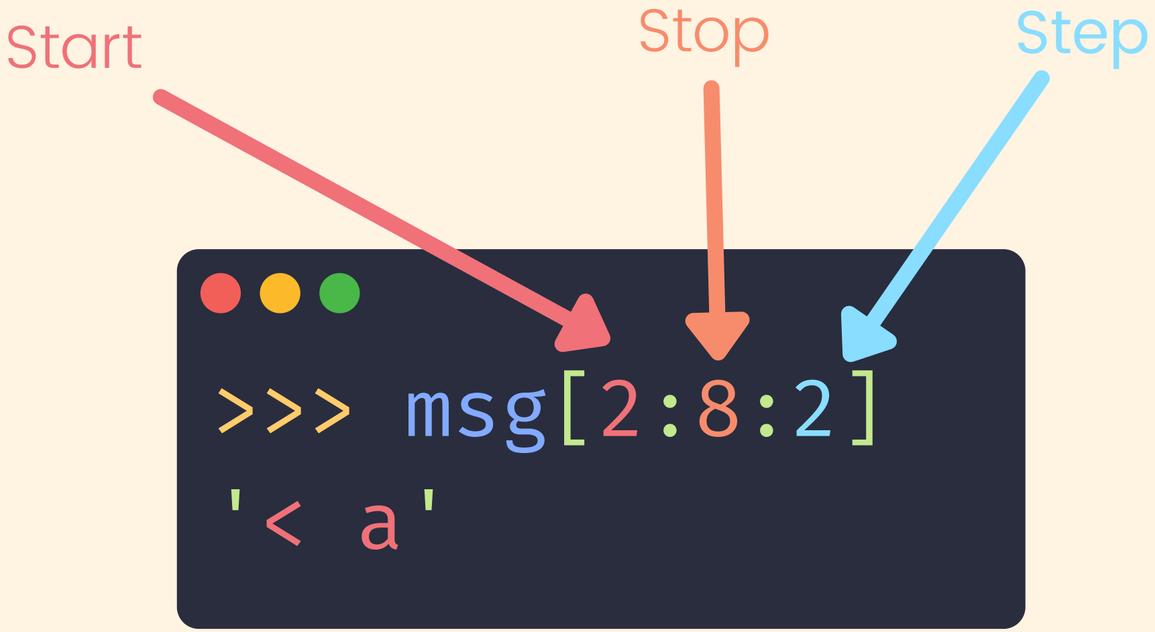

0 1 2 3 4 5 6 7 8
0 I
1
2 <
3 3
4
5 C
6 a
7 t
8 s
-9 -8 -7 -6 -5 -4 -3 -2 -1

Slices

0	1	2	3	4	5	6	7	8
I		<	3		C	a	t	s



Slices with a Step



STRINGS + FUNCTIONS

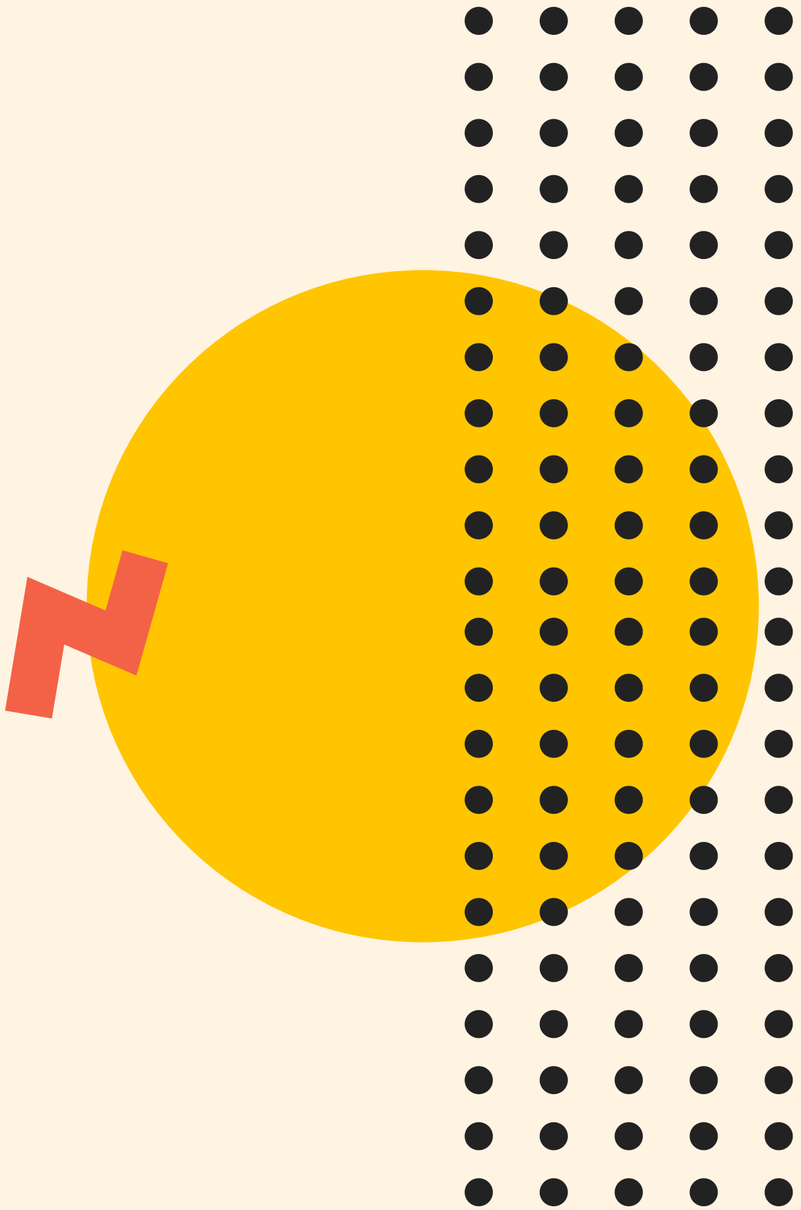
STRINGS + FUNCTIONS

Functions

FUNCTIONS ARE REUSABLE ACTIONS THAT HAVE A NAME

```
func_name()
```

TO EXECUTE A FUNCTION, WE USE PARENS ()

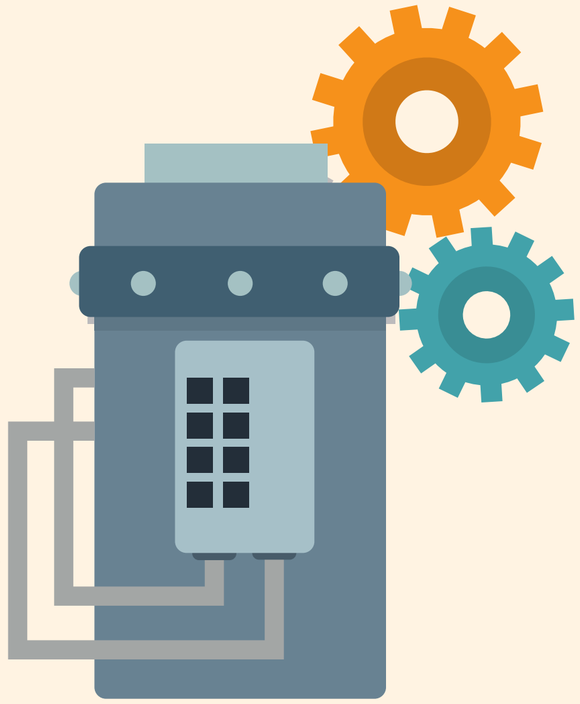


Inputs

Output

avg(20,25)

avg(3,2,5,6)

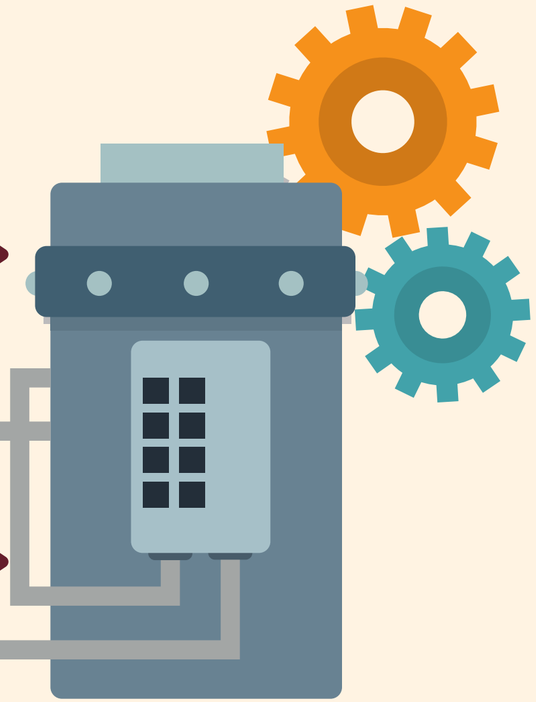


22.5

4

Inputs

Output

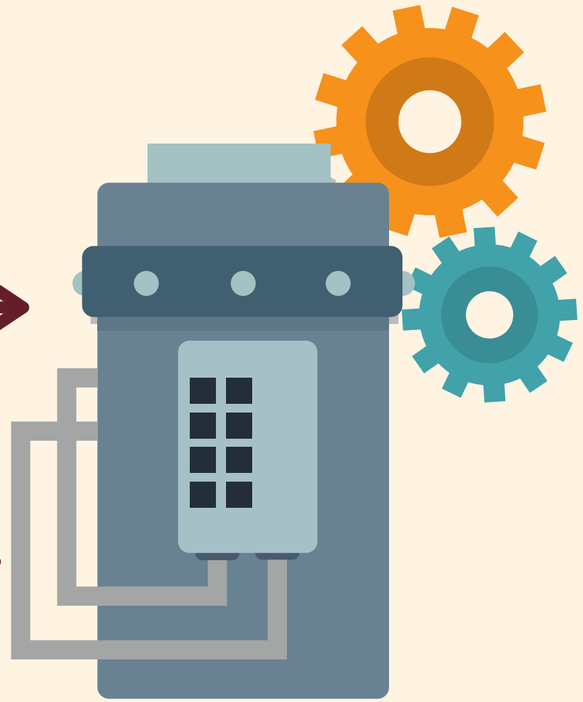
`get_weather(10003)` →  → `"23 f"`

`get_weather(92328)` → → `"78 f"`

Inputs

Output

```
login('todd', 'jjkh2fj!d')
```



False

```
login('todd', 'kittycat')
```

True

Arguments

(Fancy word for inputs)

```
>>> burger(bun,
secret_sauce, lettuce,
tomato, bacon, cheese,
well_done)
```

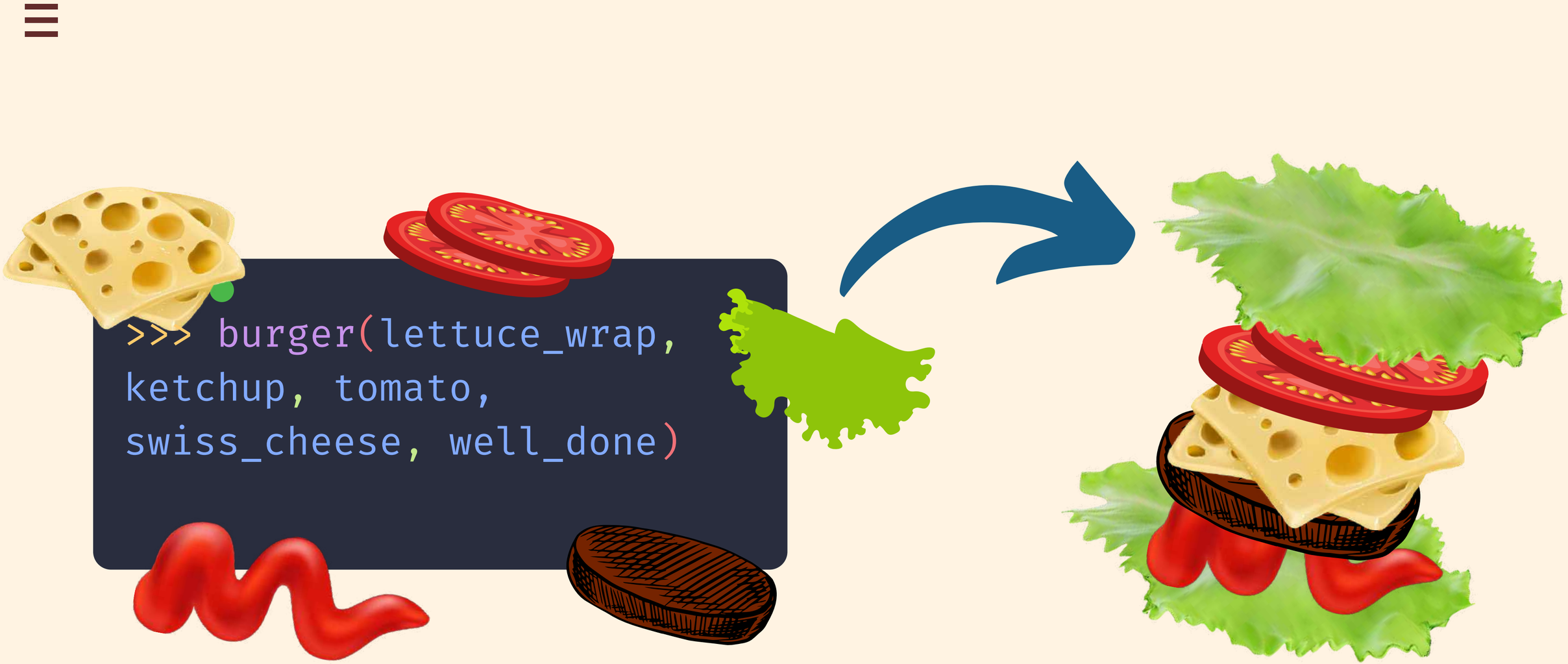
Arguments

(Fancy word for inputs)

```
>>> burger(bun, tomato,
bacon, cheese)
```


>>> burger(sesame_bun,
secret_sauce, lettuce,
tomato, bacon, cheese,
well_done)


>>> burger(sesame_seed_bun,
tomato, bacon, cheese,
well_done)

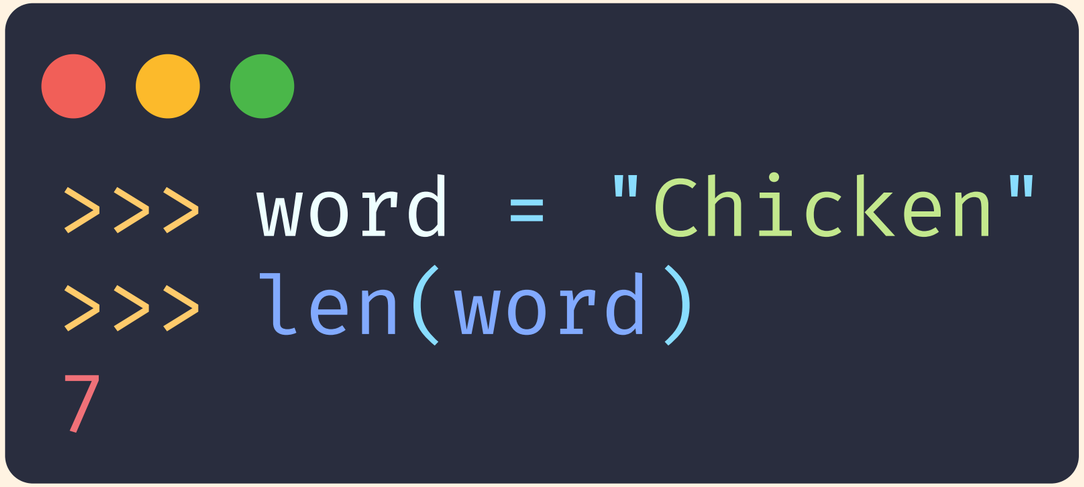

>>> burger(lettuce_wrap,
ketchup, tomato,
swiss_cheese, well_done)

```
classmethod()    isinstance()     property()
delattr()        issubclass()     range()
dict()           iter()           reversed()
dir()            len()            round()
filter()         locals()         setattr()
getattr()        map()            slice()
globals()        max()            sorted()
help()           object()         sum()
id()             open()           type()
input()          print()          zip()
```

```
classmethod()    isinstance()     property()
delattr()        issubclass()     range()
dict()           iter()           reversed()
dir()            len()            round()
filter()         locals()         setattr()
getattr()        map()            slice()
globals()        max()            sorted()
help()           object()         sum()
id()             open()           type()
input()          print()          zip()
```

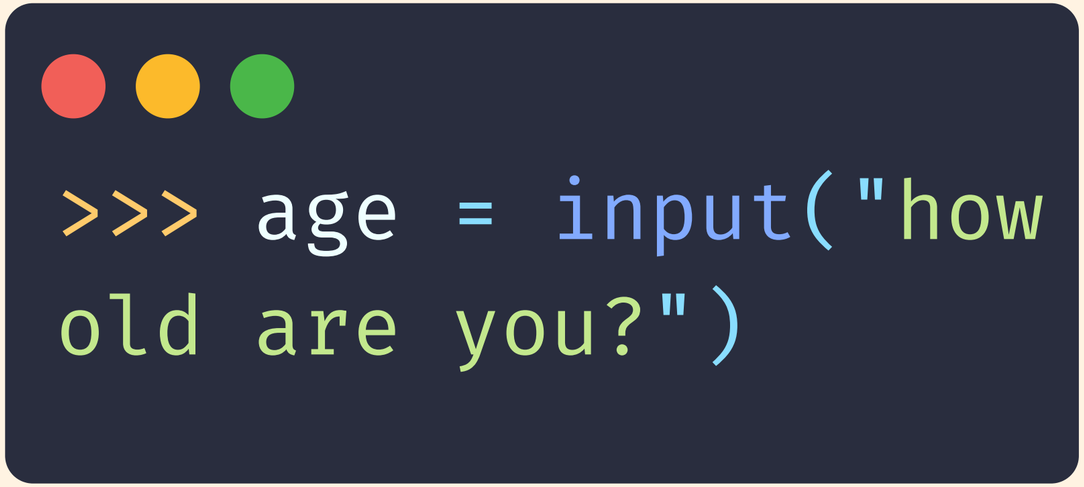
Length

The **len()** function will return the length of whatever item we pass to it. So far Strings are the only sequence we've seen, but soon we will see others!



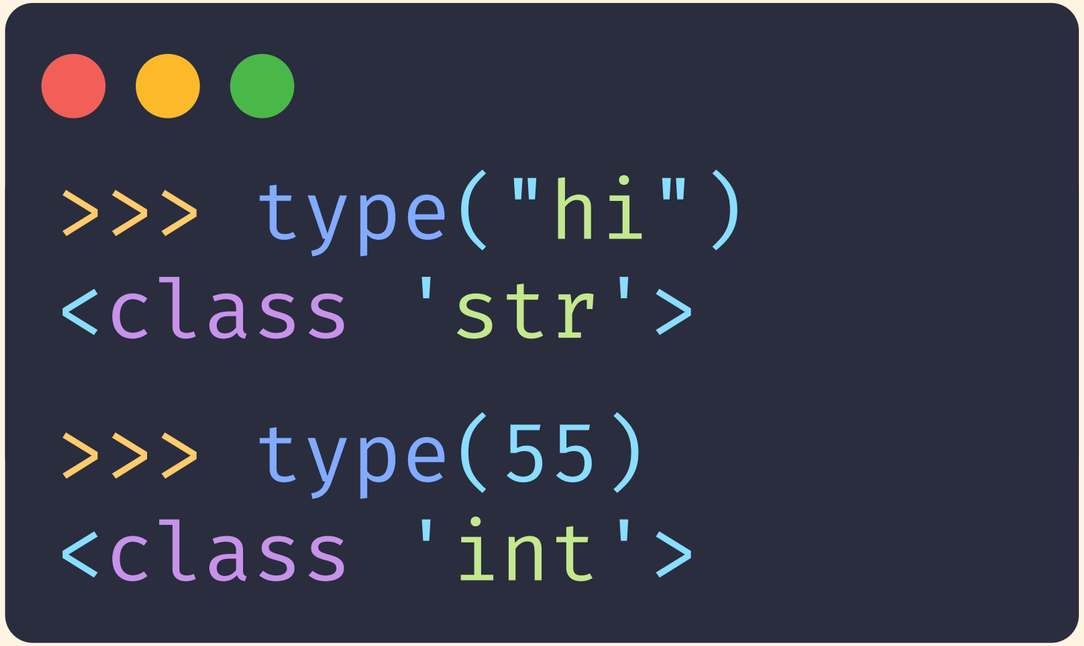
Input

The **input()** function will prompts a user to enter some input, converts it into a string, and then returns it. We can use it to gather user input in our programs



Type

The **type()** function accepts an input object and will return the type of that object



Casting Types!

```
>>> int("12")
    12
>>> float("3.3")
    3.3
>>> str(44.5)
    '44.5'
```

Print

The **print()** function prints any arguments we pass to it to "standard output". It does not return anything.

f strings

f-strings are an easy way to generate strings that contain interpolated expressions. Any code between curly braces { } will be evaluated and then the result will be turned into a string and inserted into the overall string.

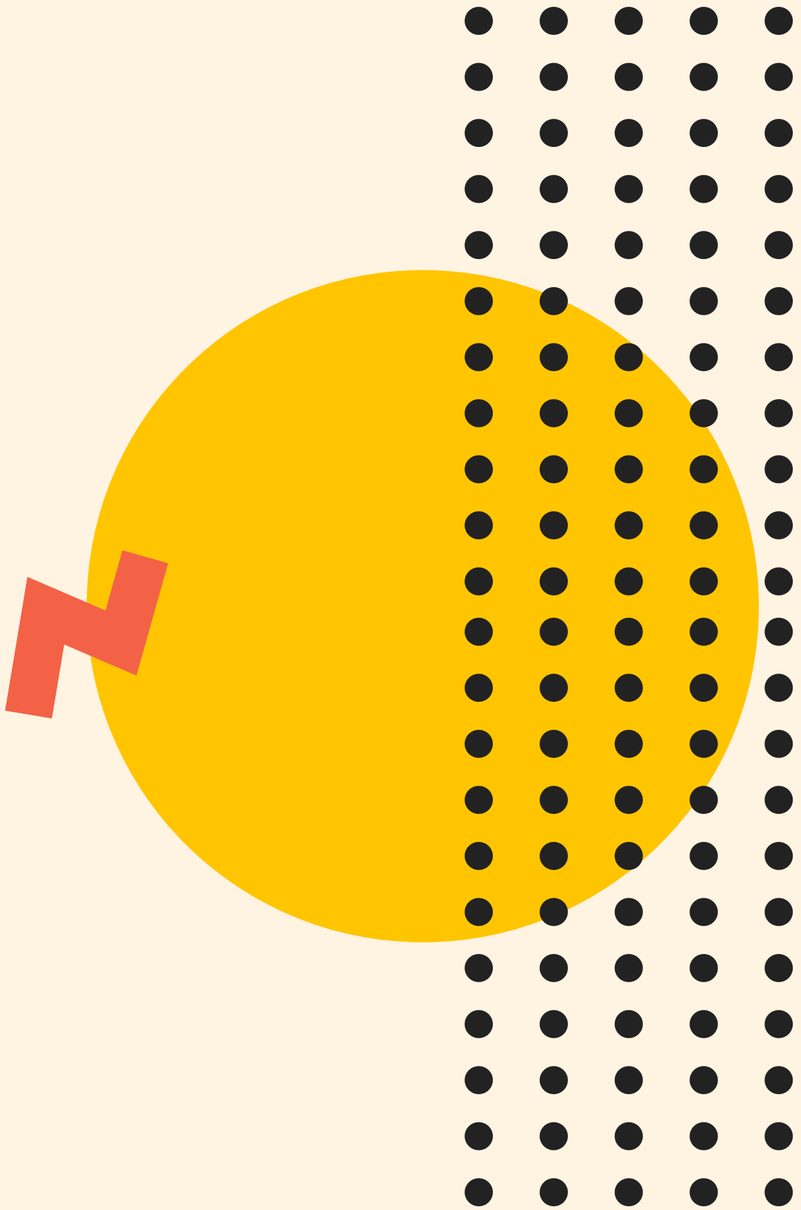
```
>>> f"there are {24*60*60} seconds in a day"
"there are 86400 seconds in a day"
```

Methods

METHODS ARE FUNCTIONS THAT "LIVE" ON OBJECTS

```
thing.method()
```

METHODS AUTOMATICALLY HAVE ACCESS TO THE OBJECT THEY ARE CALLED ON.



```
thing.method()
```

String Methods

str.capitalize()
str.endswith()
str.find()
str.join()
str.lower()
str.lstrip()
str.removeprefix()
str.removesuffix()
str.replace()
str.rfind()
str.rindex()
str.rjust()
str.split()
str.startswith()
str.strip()
str.swapcase()
str.title()
str.upper()

Capitalization Methods

```
>>> msg = "Hello world"
>>> msg.capitalize()
Hello world
>>> msg.upper()
HELLO WORLD
>>> msg.lower()
hello world
```

```
str.upper( )
```

accepts no arguments!

```
str.strip([chars])
```

chars is optional

`strip()` Strips space characters

`strip('-')` Strips '-' characters

`strip('=-')` Strips '=' and '-' characters

```
str.replace(old, new, [count])
```

old and new are required

```
str.replace(old, new, [count])
```

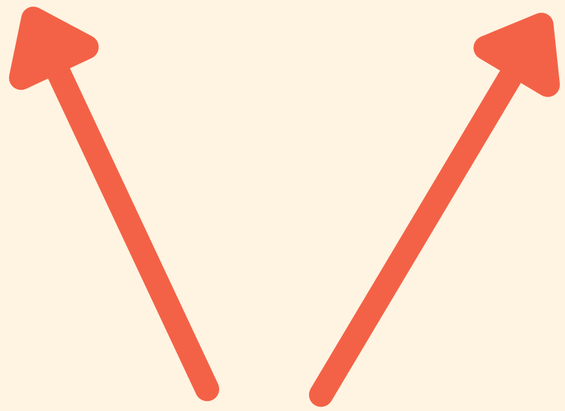
count is optional

```
find(sub[, start[, end]]) -> int
```

```
find(sub[, start[, end]]) -> int
```

sub is a required argument

```
find(sub[, start[, end]]) -> int
```



anything in [] is optional

```
find(sub[, start[, end]]) -> int
```

find returns an integer

`find("c")` returns index where "c" is first found

`find("c", 5)` returns index where "c" is first found, after index 5

`find("c", 5)` returns index where "c" is first found, after index 5

method chaining

Strip Methods

```
>>> msg = "...end..."
>>> msg.strip('.')
end
>>> msg.lstrip('.')
end...
>>> msg.rstrip('.')
...end
```

Find & Index

```
>>> msg = "Cat in a hat"
>>> msg.find('a')
1
>>> msg.rfind('a')
10
>>> msg.index('a')
1
```

Replace & Count

```
>>> msg = "Hot dog"
>>> msg.replace('o','u')
Hut dug
>>> msg.replace('o','u',1)
Hut dog
>>> msg.count('o')
2
```

BOOLEANS + COMPARISONS

Basic Data Types

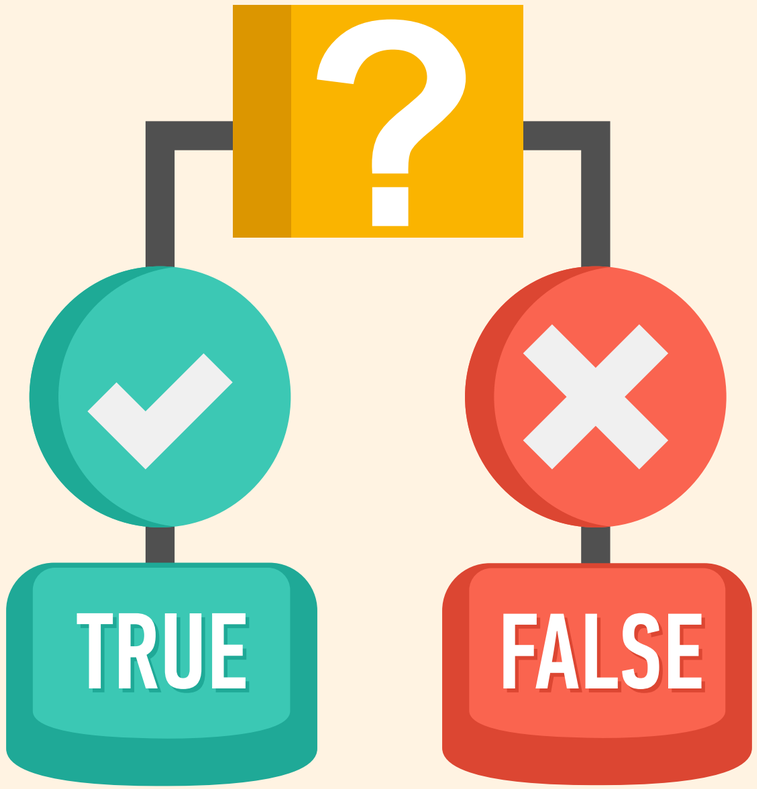
Strings

Integers

Booleans

Floats

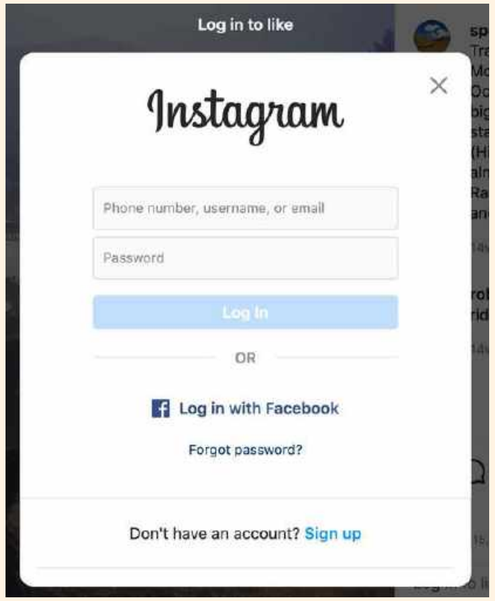
Decision Making



Is the user logged in?

no

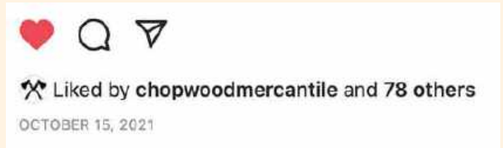
yes



Has the user already liked the photo?

no

yes



Like the photo

un-like the photo

Potential Decisions

Is the game over?

Does the user have any guesses left?

Is 'S' in the target word?

Is 'O' in the target word?

Is 'O' in the correct location?

Is 'U' in the target word?

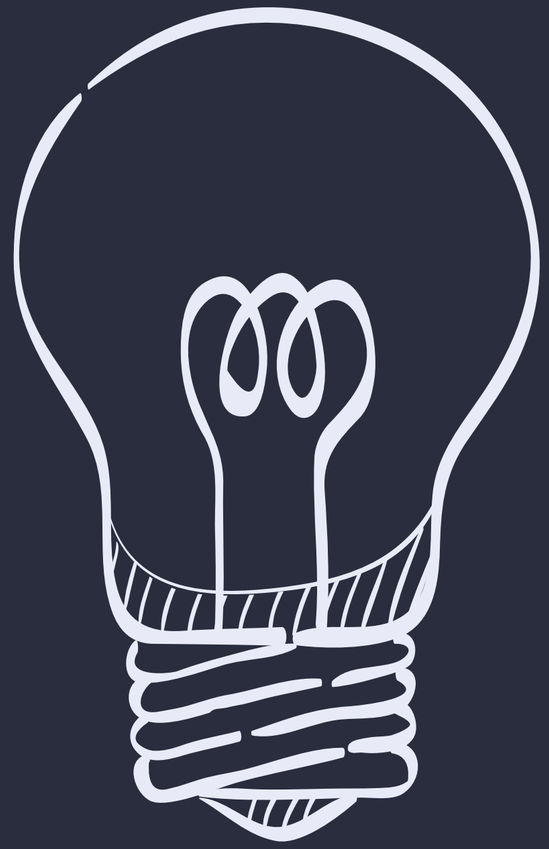
Is 'N' in the target word?

Is 'N' in the correct location?

Is 'D' in the target word?

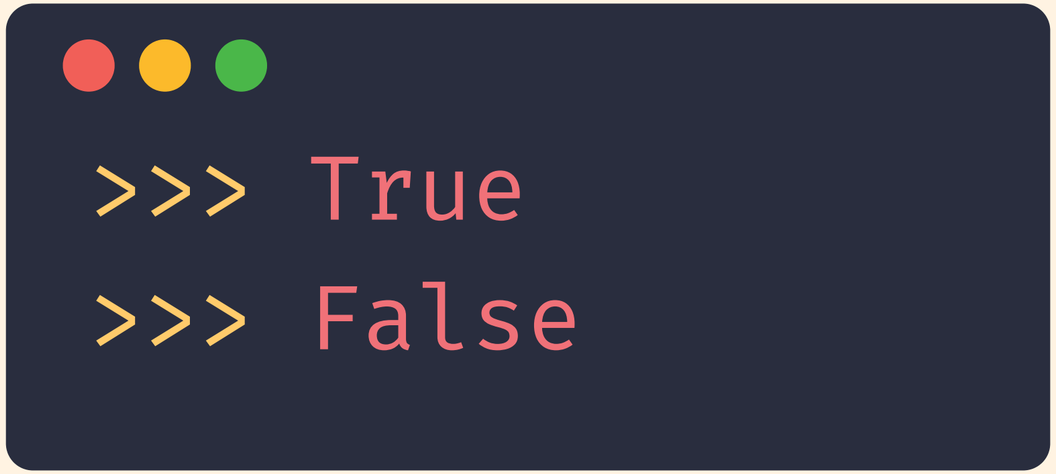
ON

OFF

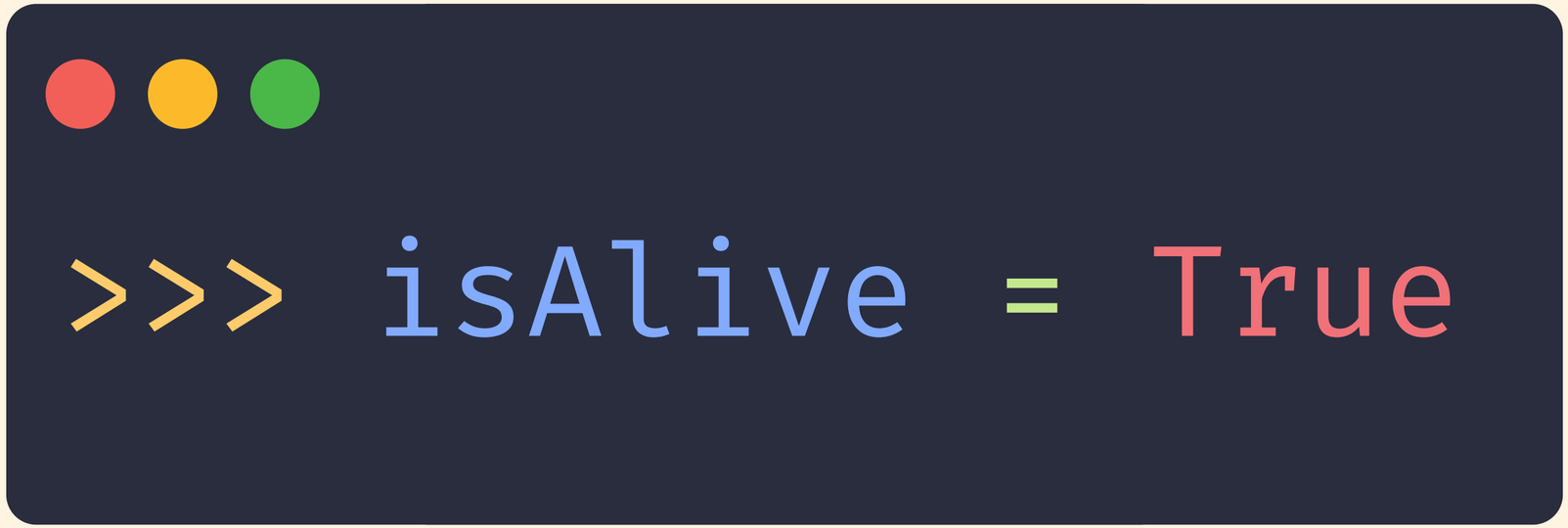


Booleans

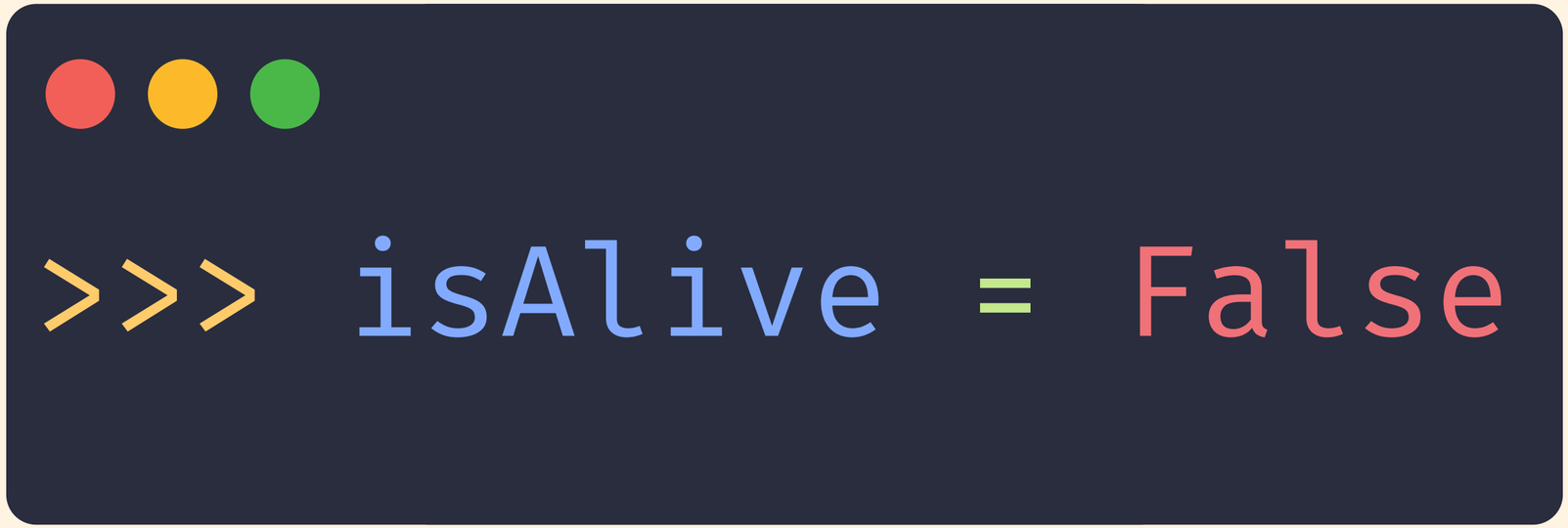
Booleans are another basic Python type. There are only two possible values: **True and False.** Notice the capitalization!!



Booleans



Booleans



Operators

Operators are special characters in Python that perform operations on value(s). Below are some of the most common:

+	*	>	<=	and	is	=	*=
-	/	<	==	or	in	+=	/=
**	%	>=	!=	not	!=	-=	|=

Comparisons

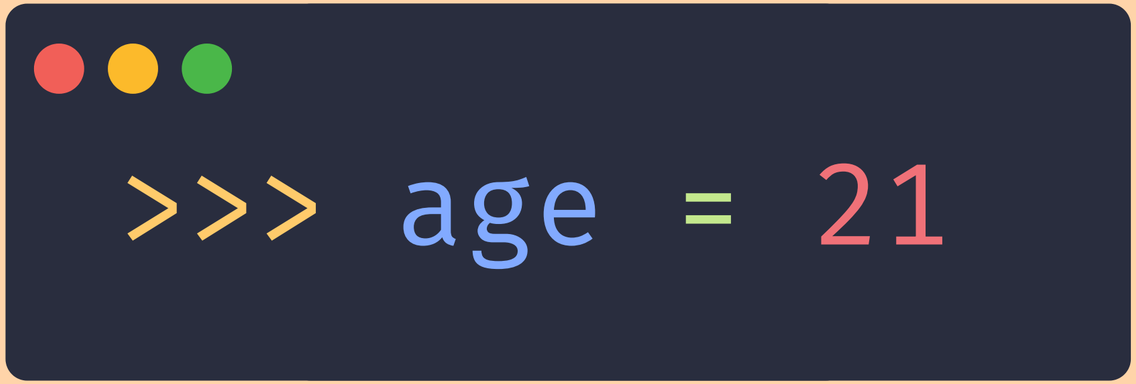
>	Greater Than
<	Less Than
>=	Greater Than Or Equal To
<=	Less Than Or Equal To

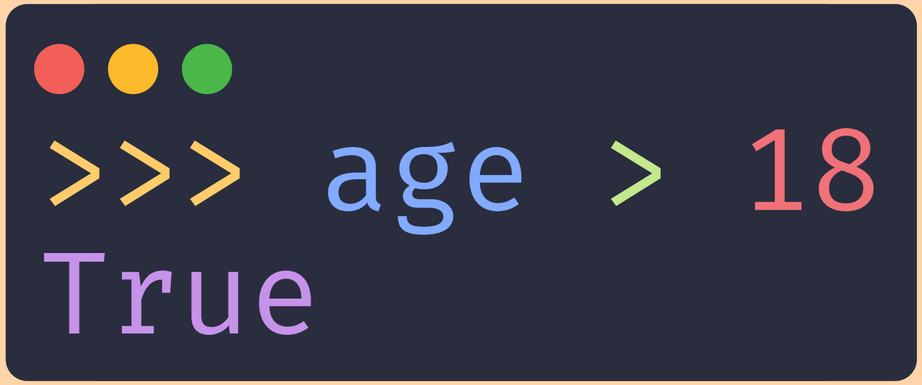
`a > b` Truthy if a is greater than b

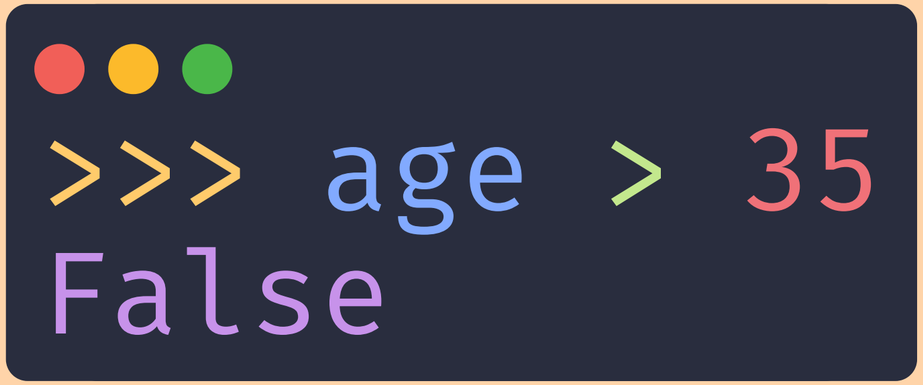
`a < b` Truthy if a is less than b

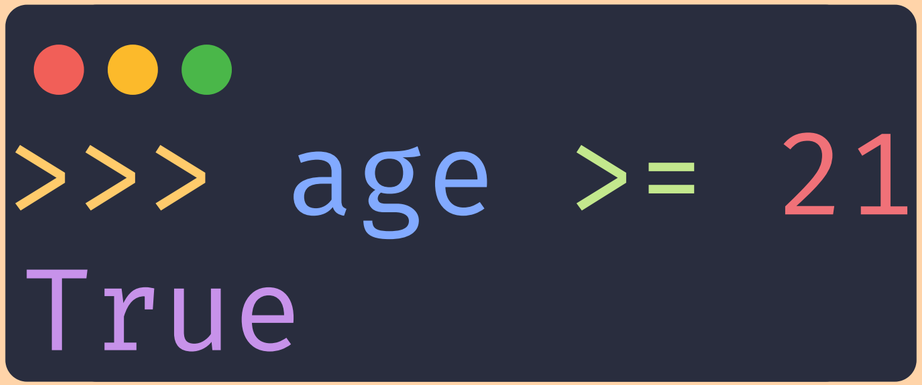
`a >= b` Truthy if a is greater than or equal to b

`a <= b` Truthy if a is less than or equal to b


>>> age = 21

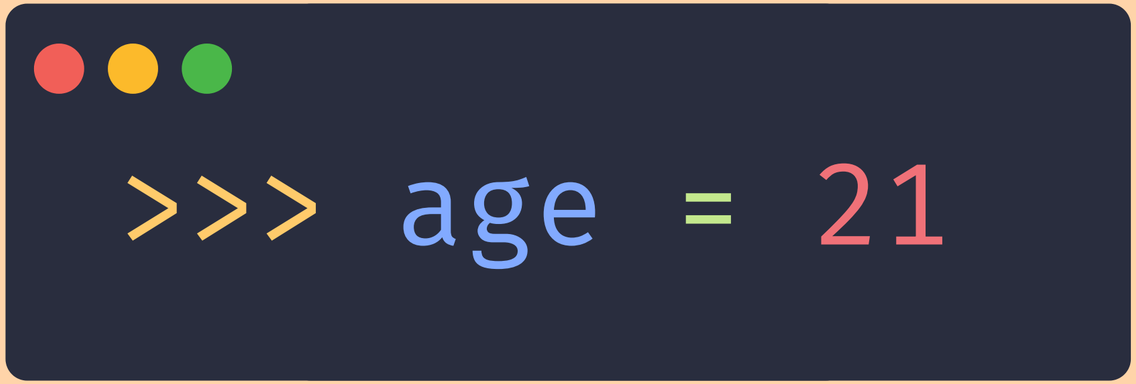

>>> age > 18
True

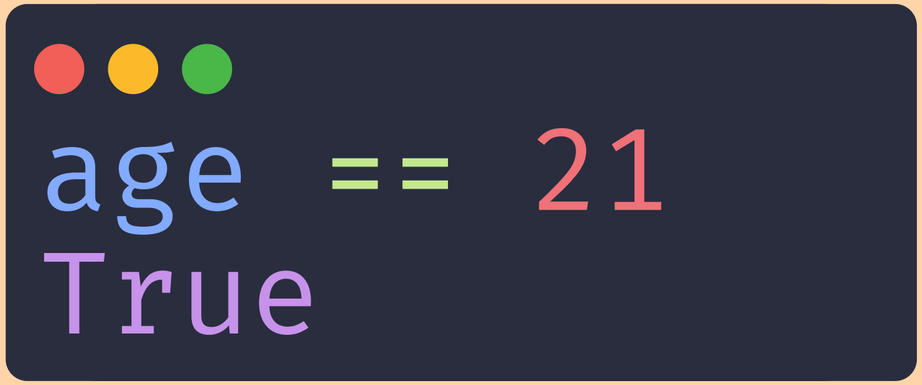

>>> age > 35
False

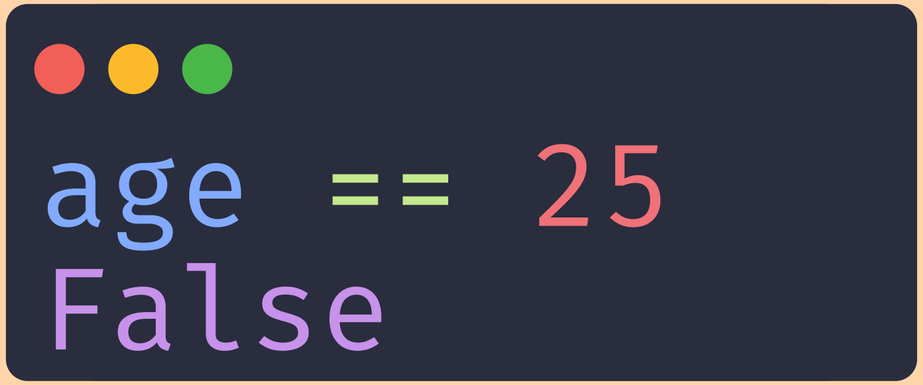

>>> age >= 21
True

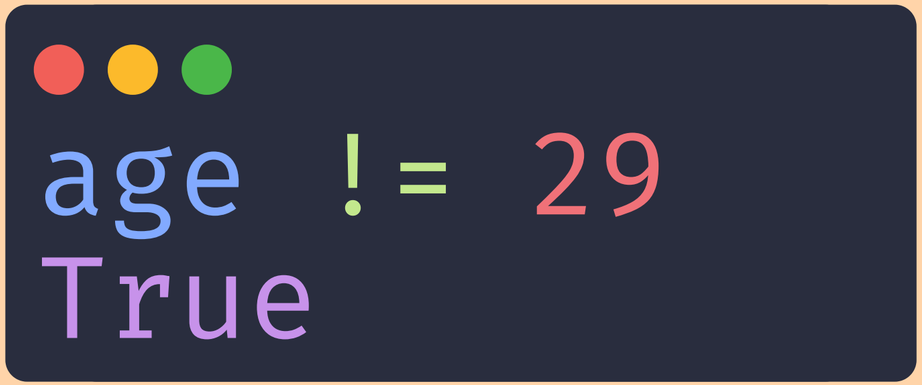
Comparisons

`==`	Equal To
`!=`	Not Equal To


>>> age = 21


age == 21
True

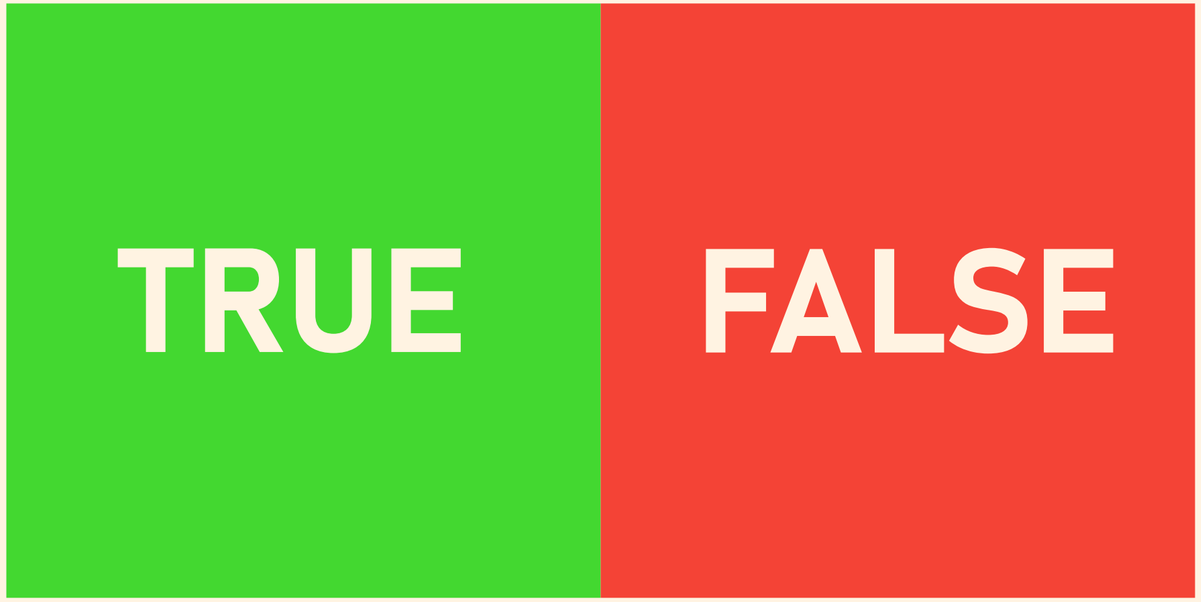

age == 25
False


age != 29
True

Identity

is	Evaluates to True if a and b both refer to the same object in memory
is not	Evaluates to True if a and b do NOT refer to the same object in memory

Every value is inherently Truth-y or False-y in Python



False-y

False

0.0

0

None

range(0)

Empty Strings:
''
""
''''''
""""""""

Empty Data Structures:
[ ]
( )
{ }
set()

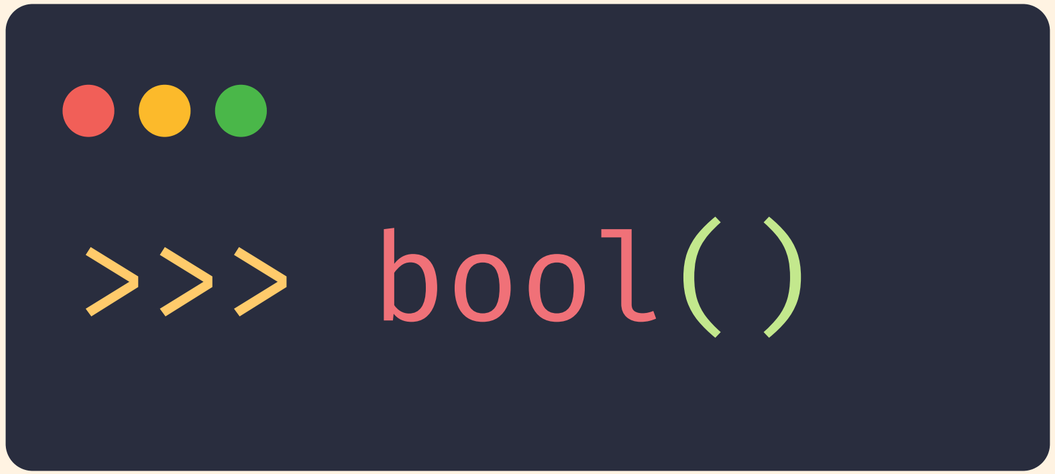
Truthy

Everything Else!

bool( )

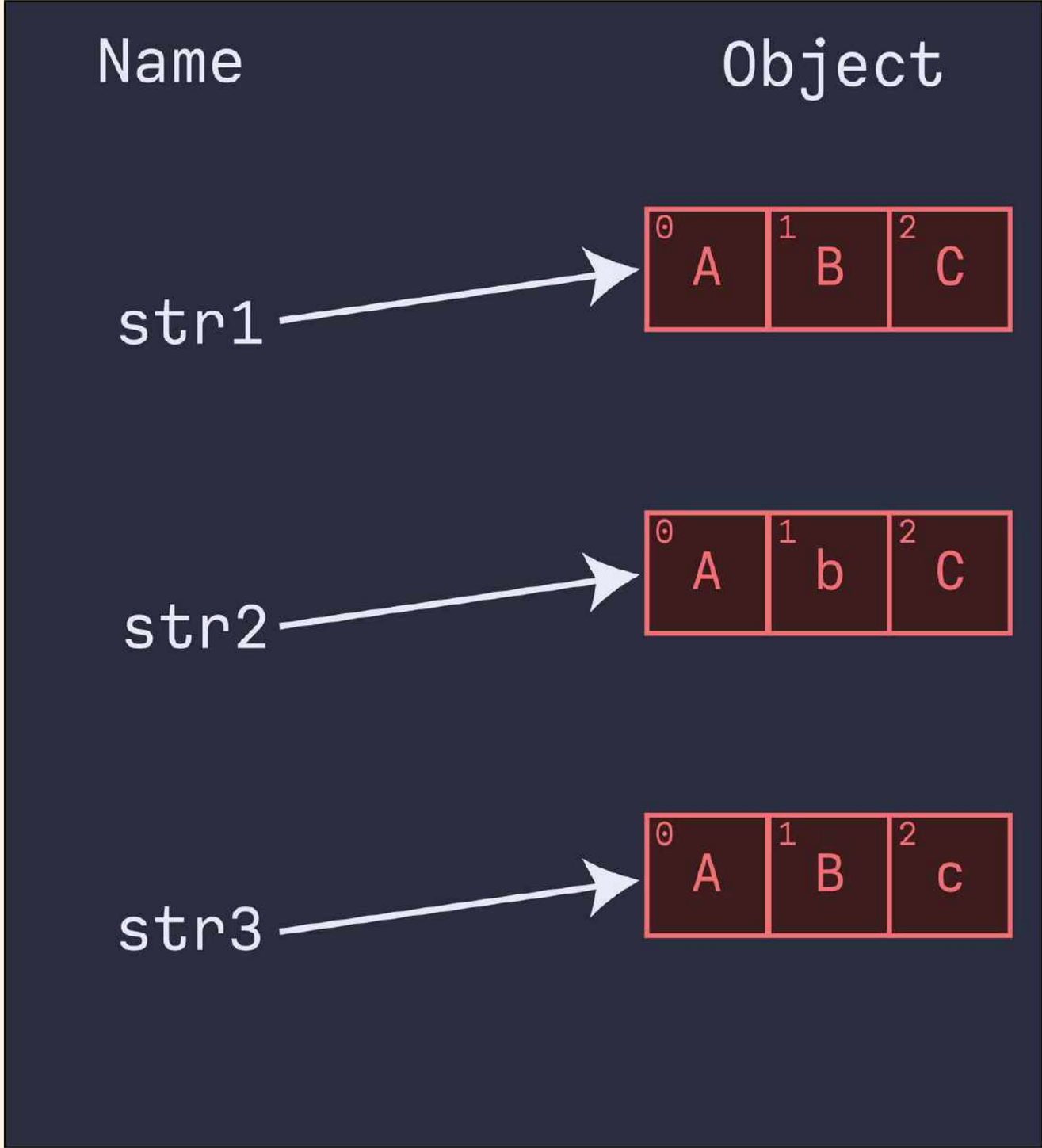
Just as we can use **int**( ), **float**( ), and **str**( ) to cast values, we can use **bool**() to cast a value to a Boolean.

This is one way to determine whether Python considers a value to be Truth-y or False-y



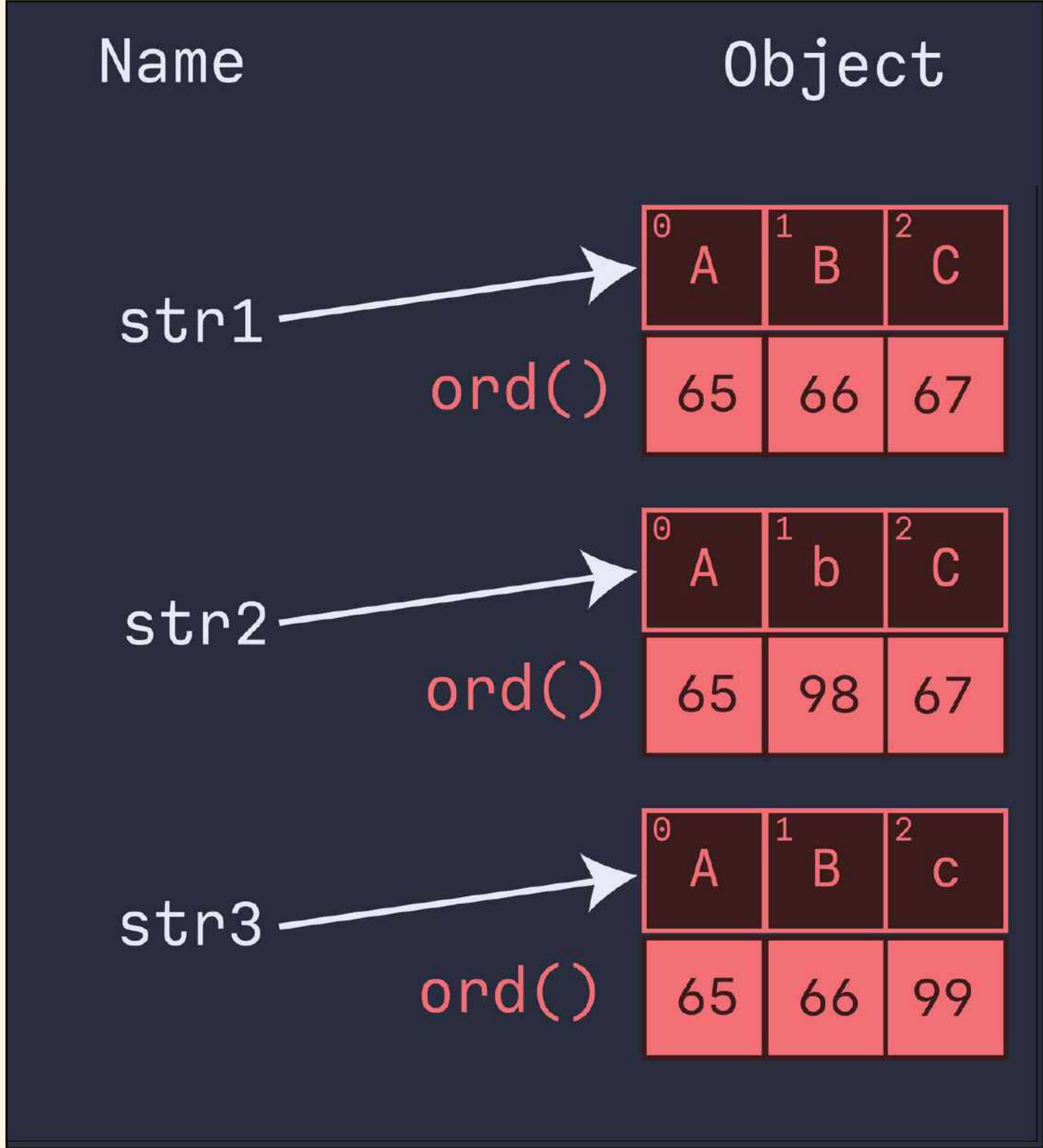
String Comparison

```
>>> str1='ABC'

>>> str2='AbC'

>>> str3='ABc'
```



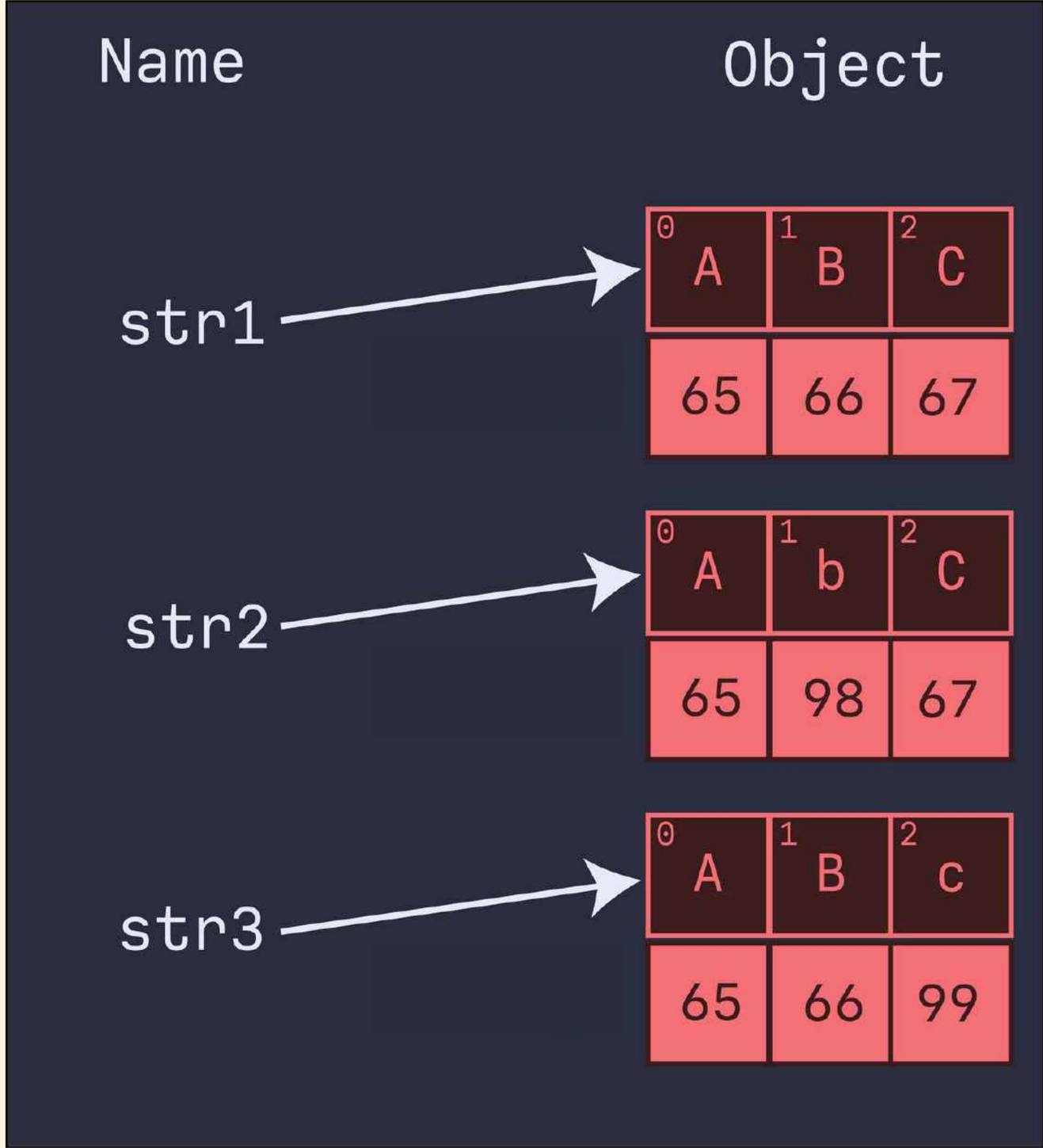
String Comparison

```
>>> ord('A')
65
>>> ord('B')
66
>>> ord('C')
67
>>> ord('b')
98
>>> ord('c')
99
```



String Comparison

```
>>> str1 > str2
False

>>> str2 > str3
True

>>> str1 > str3
False
```



logical and

The **and** operator will evaluate to True only if both the left and right sides evaluate to True.

```
'a' == 'a' and 1 < 5
True
```

logical and

The **and** operator will evaluate to True only if both the left and right sides evaluate to True.

```
'a' == 'a' and 1 < 5
True
```