Why Python?

Versatile

Data science, machine learning, web development, & more

Strong Community

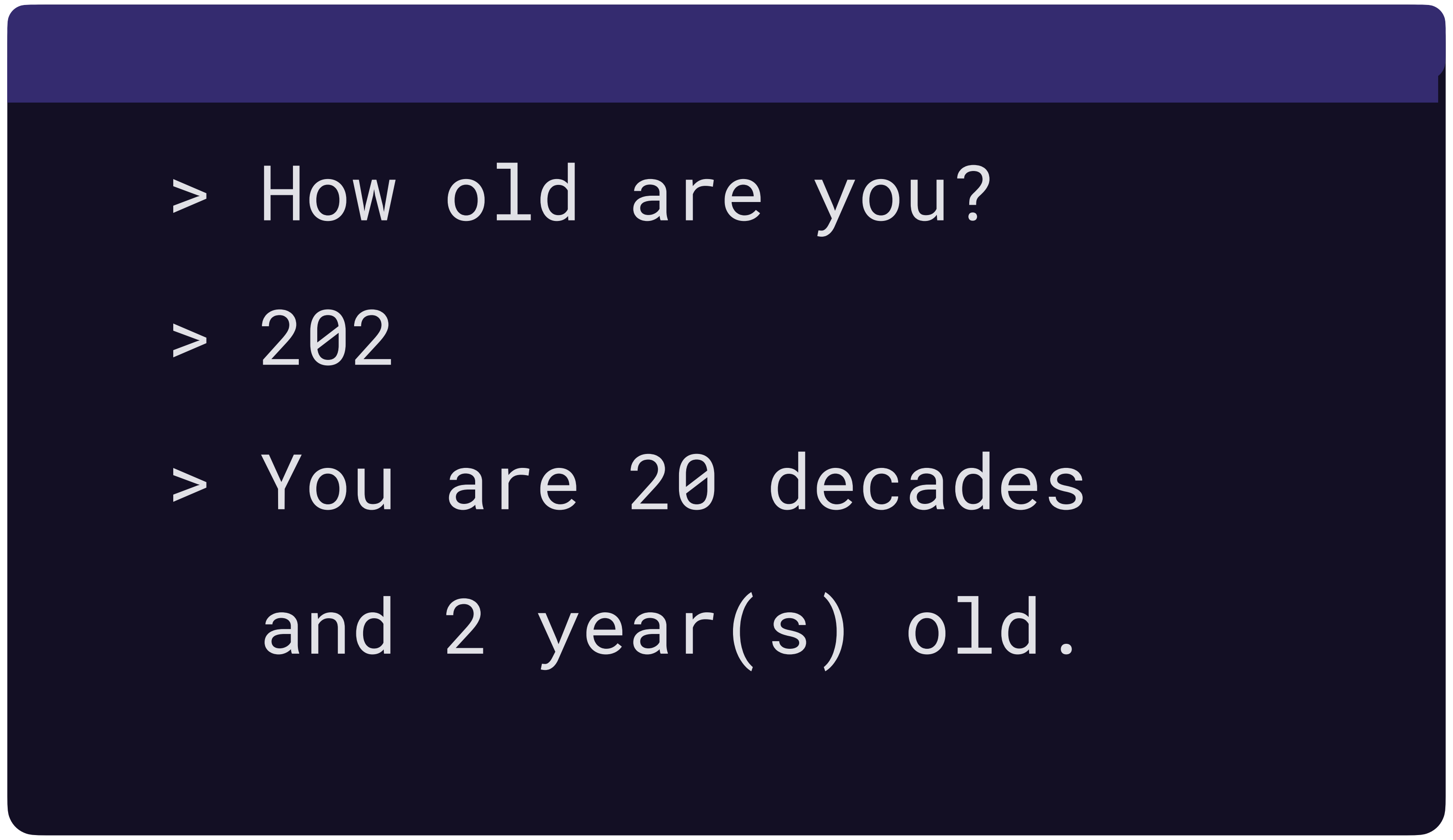
There's a package for everything

```
name = "IBM"
if name == "IBM":
    print("Hi IBM!")
else:
    print("Imposter!")
```

Easy to Learn

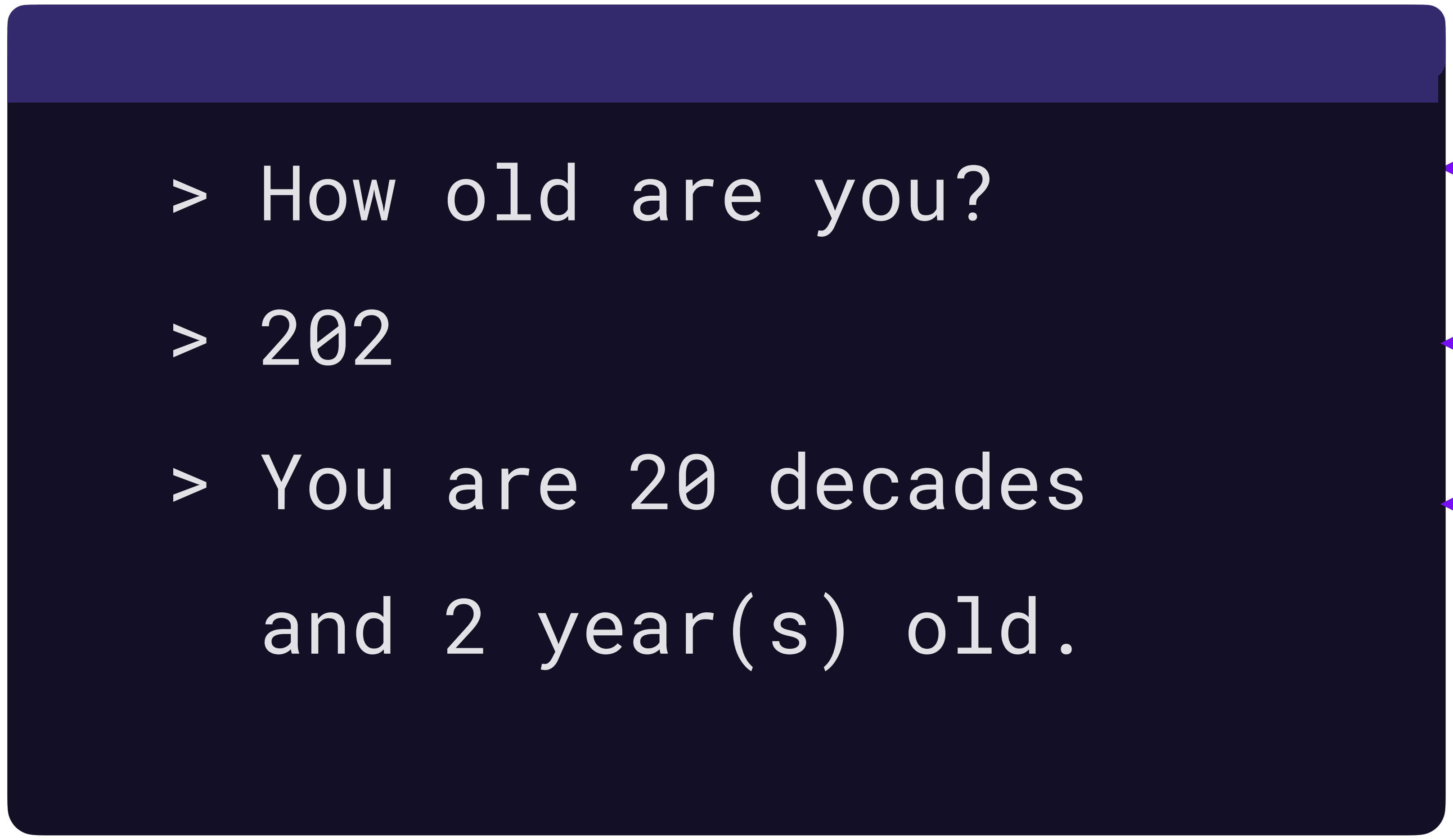
Easy-to-read, concise, interpreted language

Where Do We Start?



```
> How old are you?
> 202
> You are 20 decades
  and 2 year(s) old.
```

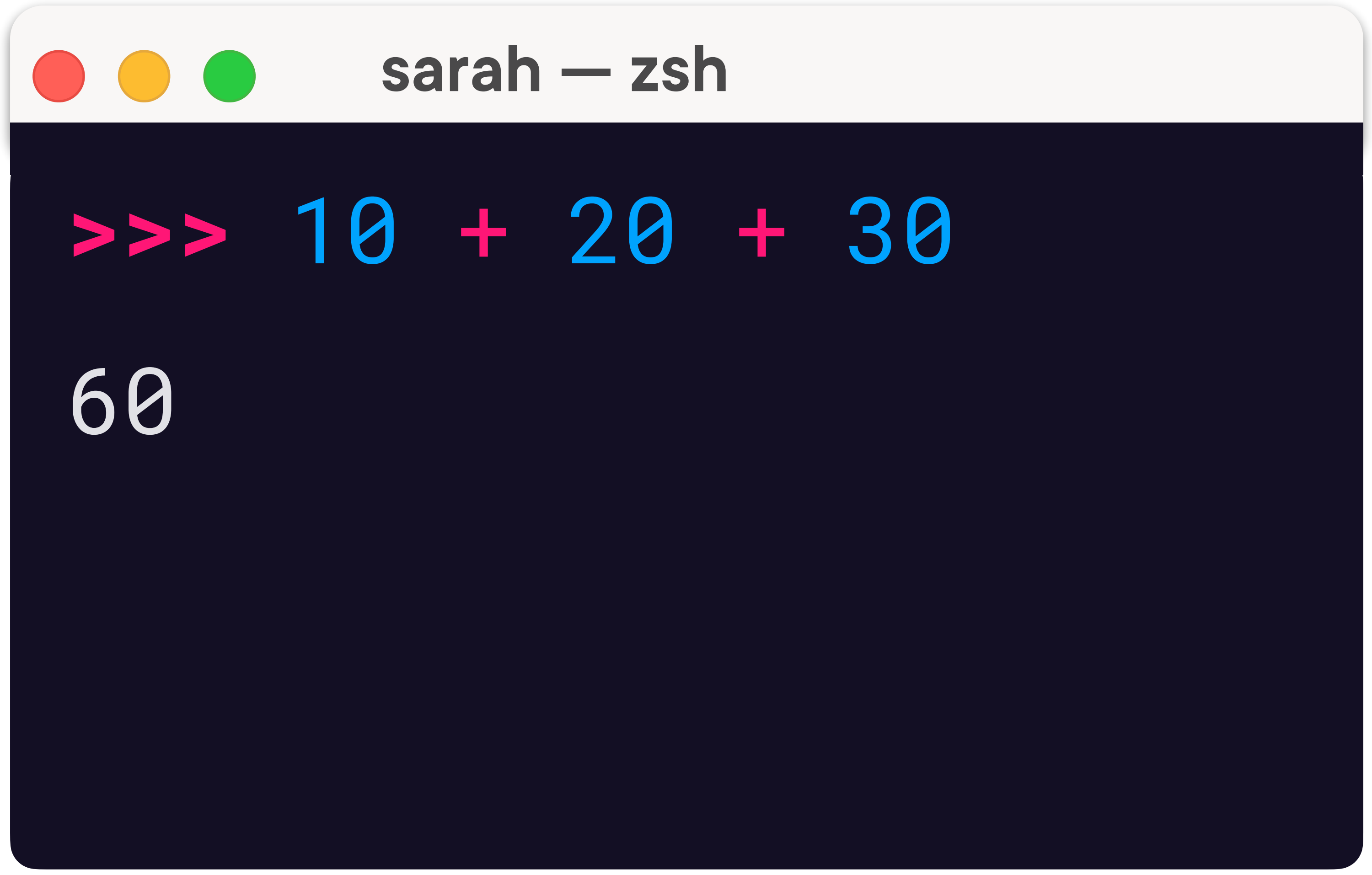
Where Do We Start?



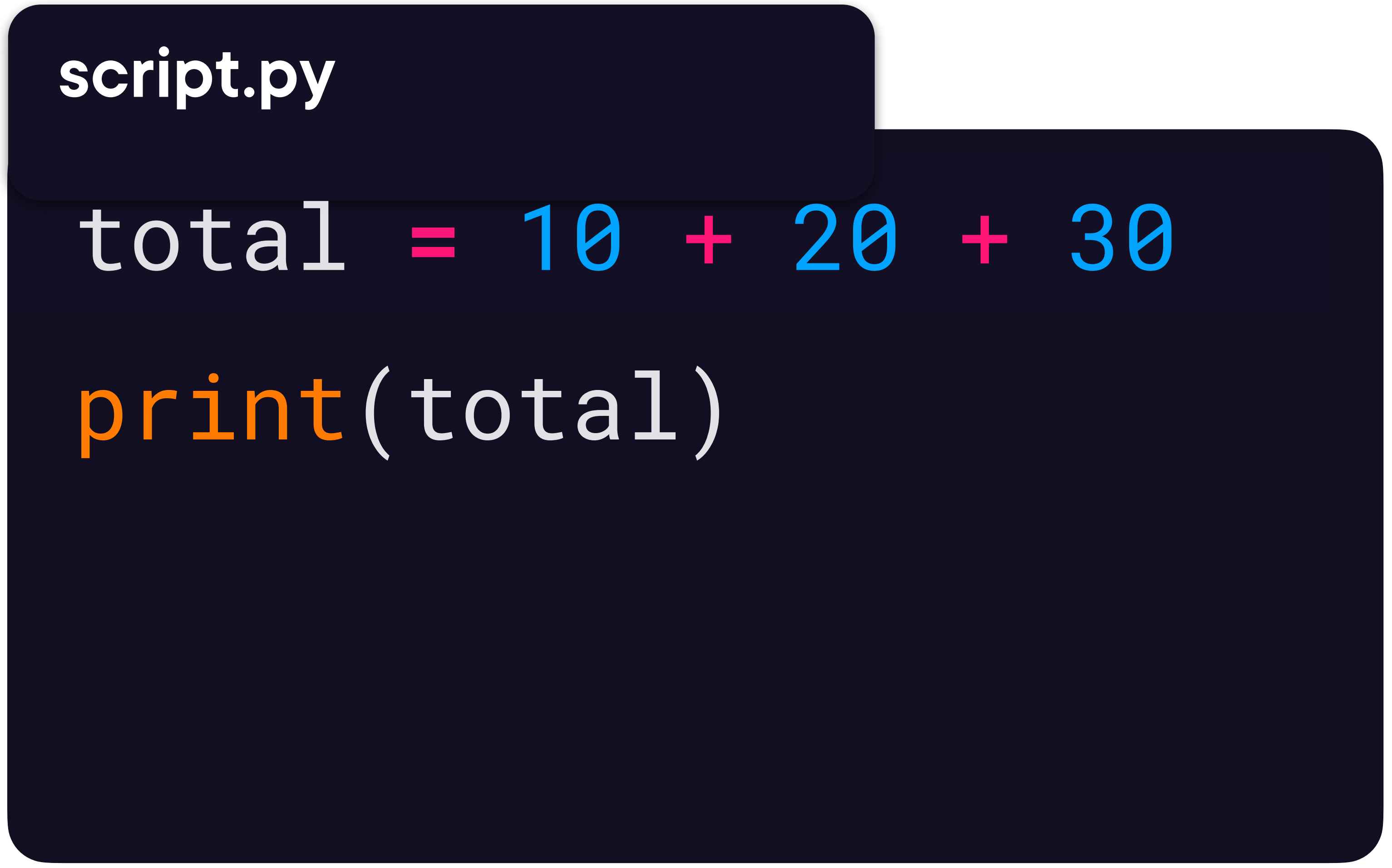
```
> How old are you?
> 202
> You are 20 decades
  and 2 year(s) old.
```

Ask the user for input
Save the input to a variable
Calculate the decades and years
Convert these numbers to text
Print the result to the screen

Where Do We Write Python Code?



```
>>> 10 + 20 + 30
60
```



```python
total = 10 + 20 + 30
print(total)
```

The Python Interactive Shell

The Python shell let's you run Python lines of code one at a time

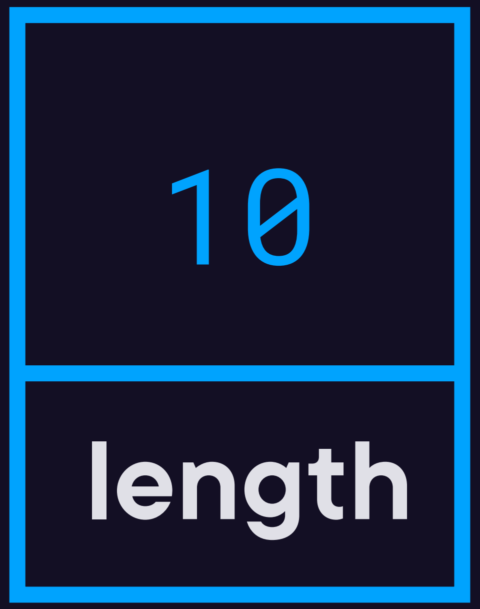
A Python File

A Python script or file is where you create longer Python programs

Saving Numbers to Variables

Assigning the value `10` *to the variable* `length`

```
>>> length = 10
```

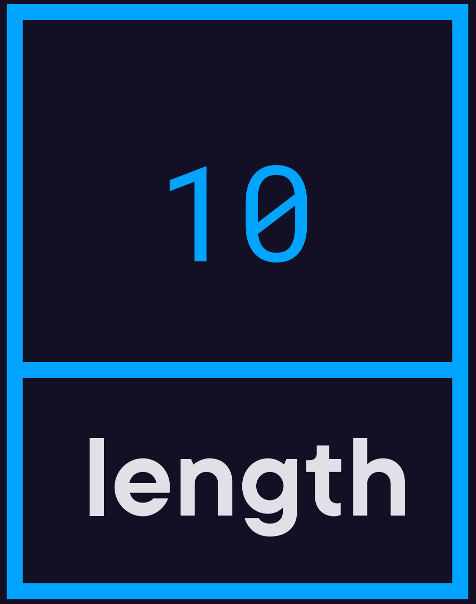


Now on your computer there is a piece of memory labeled `length` *that stores the value* `10`

Saving Numbers to Variables

```
>>> length = 10
>>> length
10
```

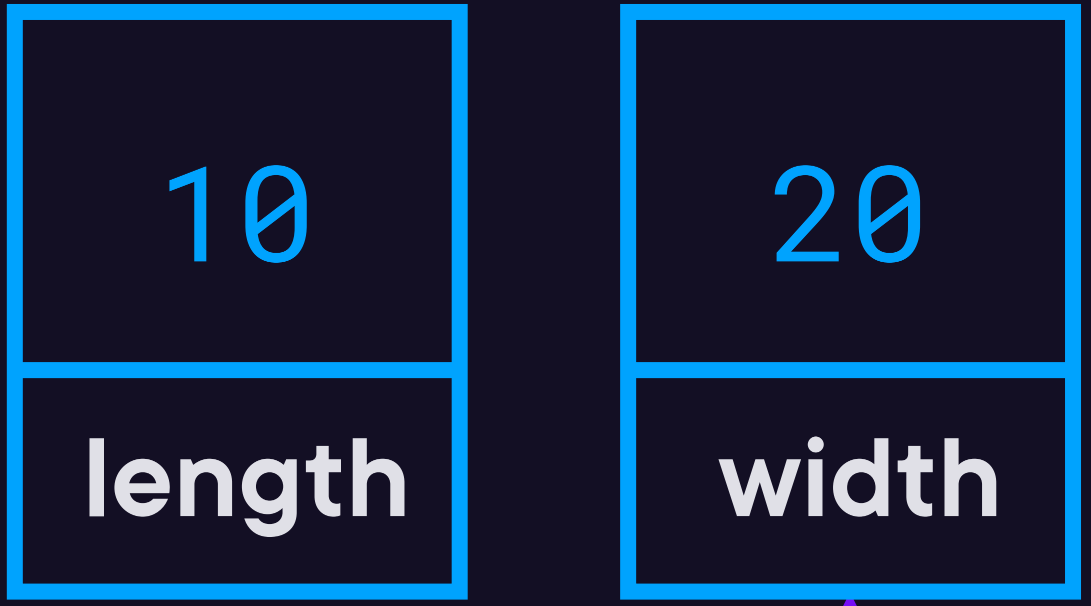
From the shell we can enter the name of the variable `length` *to see it's value and see that it's actually* `10`



Saving Numbers to Variables

```
>>> length = 10
>>> width = 20
```

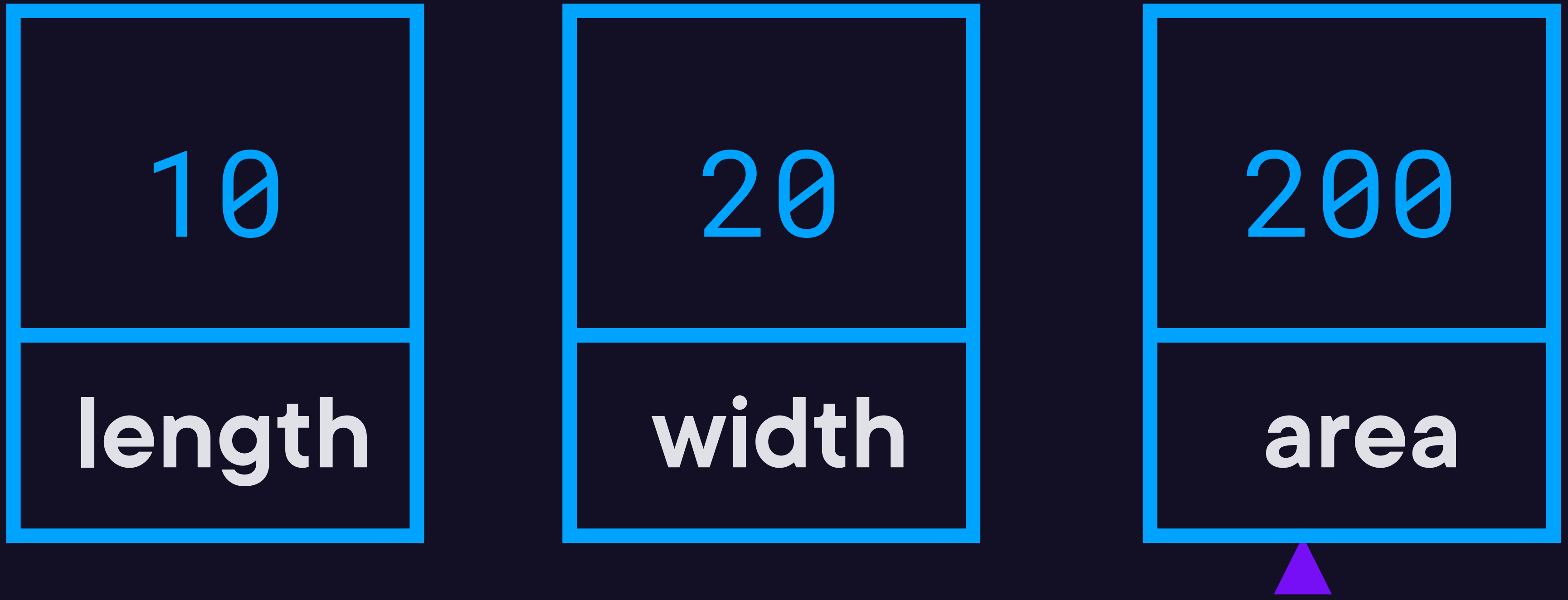
Let's also add the width of the rectangle



Now we have another variable stored in memory

Saving Numbers to Variables

```
>>> length = 10
>>> width = 20
>>> area = length * width
```



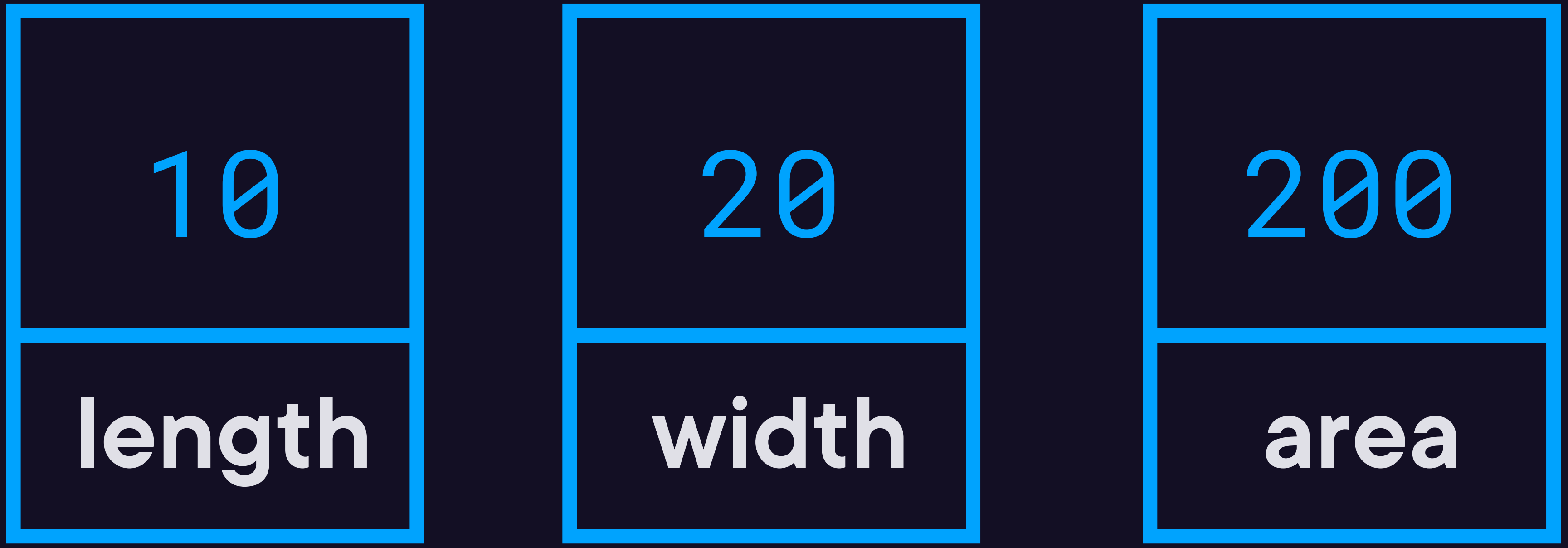
Now we can calculate the area with the multiplication operator

And now we have another variable stored in memory

The arithmetic operators in Python are mostly the same ones you know already from a calculator: + − * /

Saving Numbers to Variables

```
>>> length = 10
>>> width = 20
>>> area = length * width
>>> area
200
```



The value of `area` *is output to the screen*

Primitive Data Types

Python assumes the type of variable based on the assigned value

```
>>> amount = 10
```

```
>>> amount = 10.50
```

int

Python infers that `amount` is an `int` since it is a whole number

float

Python infers that `amount` is a `float` since it is a decimal

A Python Script

sales_tax.py

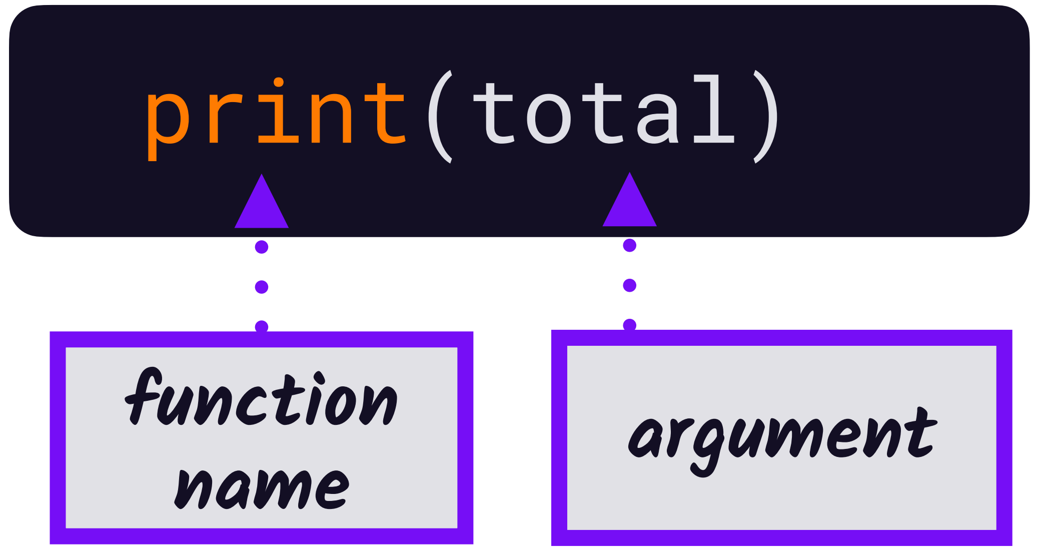
```python
amount = 10
tax = .06
total = amount + amount*tax
print(total)
```

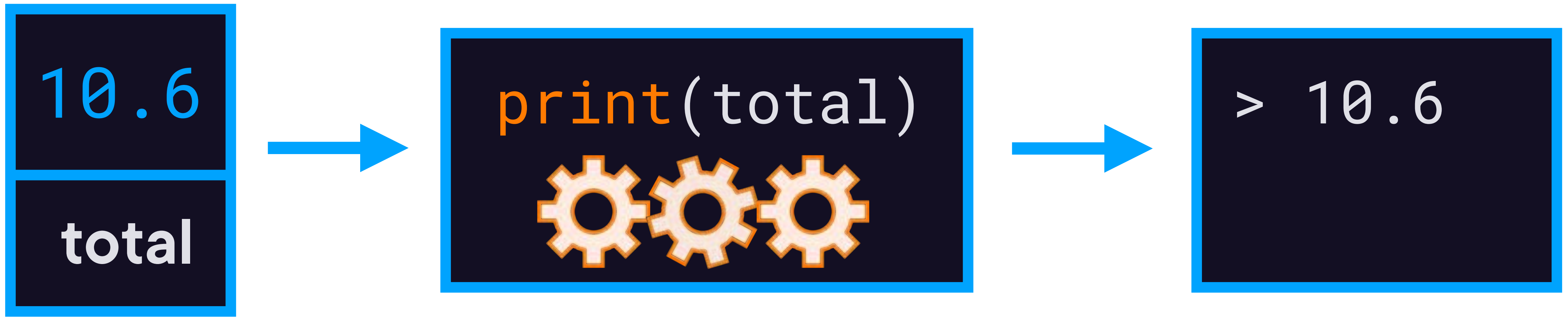
We can call the `print()` function to output `total`

```
> python3 sales_tax.py
10.6
```

Now the value of `total` is printed to the screen

Python `print()` Function





You can think of this function as a black box machine. We don't know how it works...

But we do know it will output the given value to the screen.

Data Type Conversion Functions

What if we want to convert a `float` to an `int`?

```
>>> amount = int(10.6)
>>> amount
10
```

`int()`

Use the `int()` conversion function

What if we want to convert an `int` to a `float`?

```
>>> amount = float(10)
>>> amount
10.0
```

`float()`

Use the `float()` conversion function

A String Stores Text

greeting.py

```
name = 'Sarah'

print(name)
```

Creating a String *with single quotes*

The string 'Sarah' *is saved to the variable* name

```
> python3 greeting.py
Sarah
```

The value of name *prints without quotes*

The quotes are only used to tell Python that anything inside them is a String.

Create Strings with Single or Double Quotes

greeting.py

```
store_name = "IBM's Store"
print(store_name)
```

Double quotes are useful if a single quote is literally part of the `String`

```
store_name = 'IBM's Store'
print(store_name)
```

This would cause an error because the second single quote would end the `String` *and Python doesn't know what to do with the rest.*

String Concatenation

greeting.py

```
hello = "Hello"
name = "IBM"
greeting = hello + name
print(greeting)
```

Concatenate two Strings with a +

```
> python3 greeting.py
HelloIBM
```

Notice how the two strings are smushed together? We need a space between them.

Fixing Our Program

greeting.py

```
hello = "Hello"
name = "IBM"
greeting = hello + " " + name
print(greeting)
```

Concatenate a space

```
> python3 greeting.py
Hello Sarah
```

Fixed

Fixing Our Program

greeting.py

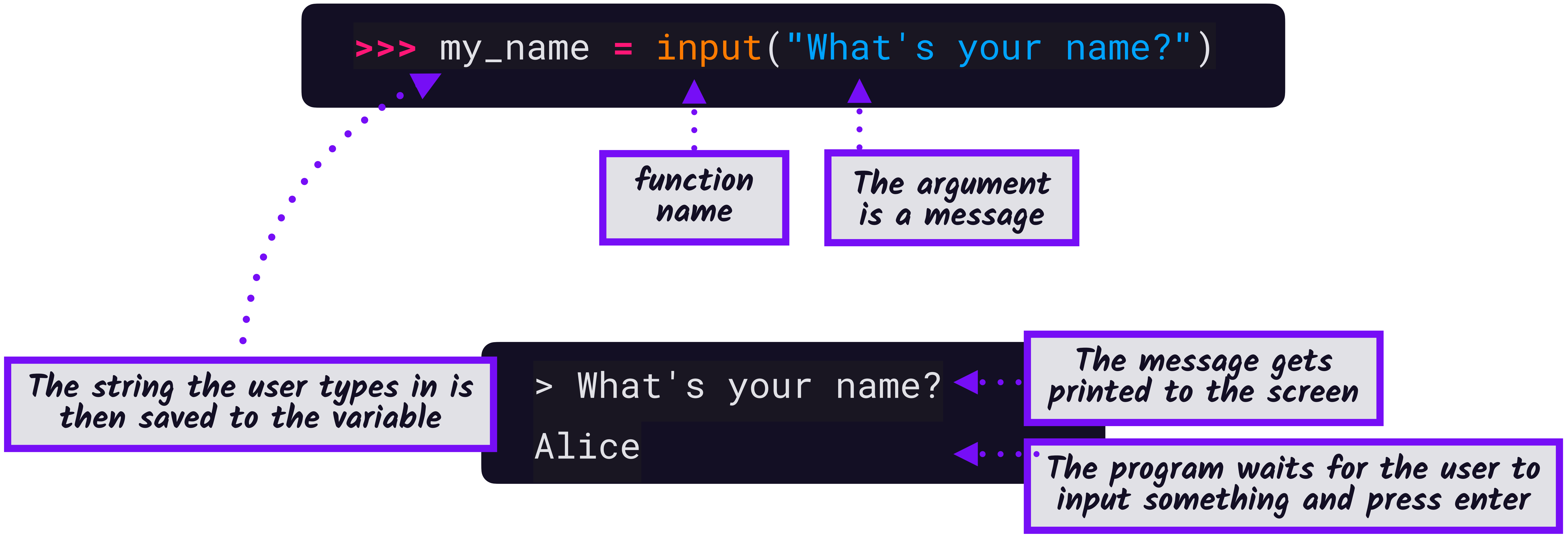
```
hello = "Hello"
name = "IBM"
greeting = hello + " " + name
print(greeting)
```

Let's ask the user for their name.

```
> python3 greeting.py
Hello Sarah
```

How can we customize this program for other names?

Python input() Function



```
>>> my_name = input("What's your name?")
```

function name
The argument is a message
The string the user types in is then saved to the variable
The message gets printed to the screen
The program waits for the user to input something and press enter

```
> What's your name?
Alice
```

Console Input

greeting.py

```
hello = "Hello"
name = input(“What's your name?”)
greeting = hello + " " + name
print(greeting)
```

input() *prints the statement, then waits for a value from the console*

```
> python3 greeting.py
What’s your name?Bob
Hello Bob
```

Notice how the name Bob is now printed inside of the greeting.

Console Input

greeting.py

```python
hello = "Hello"
name = input(“What's your name?”)
greeting = hello + " " + name
print(greeting)
```

```
> python3 greeting.py
What’s your name?Bob
Hello Bob
```

This looks bad. Can we enter the name on the next line?

Console Input

greeting.py

```
hello = "Hello"
name = input("What's your name?\n")
greeting = hello + " " + name
print(greeting)
```

\n *is a special character for a new line*

```
> python3 greeting.py
What's your name?
Bob
Hello Bob
```

Now input is entered on the next line.

Summary of Primitive Data Types

```
>>> amount = 10
```

int

```
>>> amount = 10.50
```

float

```
>>> name = "IBM"
```

string

Summary of Input and Output

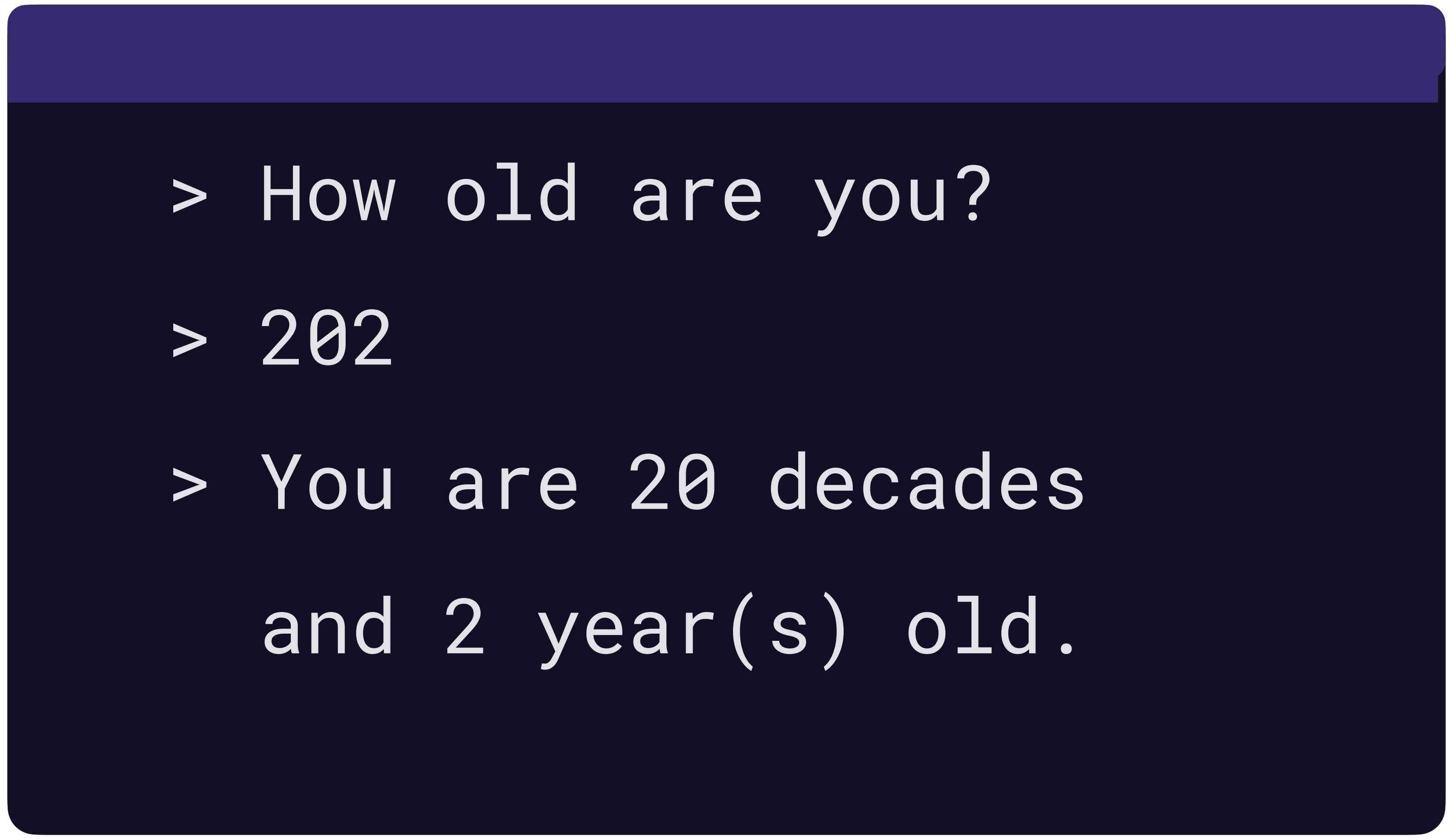
```
>>> name = input(“What's your name?\n”)
What’s your name?
IBM
```

input

```
>>> print(“Hello ” + name + "!!")
Hello IBM!!
```

output

Age Calculator



```
> How old are you?
> 202
> You are 20 decades
  and 2 year(s) old.
```

- Ask the user for input
- Save the input to a variable
- Calculate the decades and years
- Convert these numbers to text
- Print the result to the screen

How Do We Make Decisions in a Program?

A conditional statement, or if statement, lets us make decisions in Python

If it’s sunny and 90° and higher → Stay inside!

If it’s raining → Stay inside!

Otherwise → Go outdoors!

The 6 Python Comparators

<	<=	==	>=	>	!=
less than	less than equal to	equal	greater than equal to	greater than	NOT equal

Assigning 95 *to the* temp *variable*

```
>>> temp = 95
>>> temp == 95
True
```

Making a comparison is like asking the question:

Is the temp *equal to* 95*?*

Notice: the assignment is 1 = sign
And the equals to comparator is 2 == signs

The 6 Python Comparators

<	<=	==	>=	>	!=
less than	less than equal to	equal	greater than equal to	greater than	NOT equal

```
>>> temp = 95
>>> temp == 95
True

>>> temp < 90
False
```

Is the temperature less than 90*?*

An if statement

Lets us *decide* what to do: *if True*, then *do* this.

weather.py

```
temperature = 95

if temperature > 80:

```

Assign 95 *to the* temperature *variable*

Is the temperature greater than 80*?*

If this is True, *let's add something to do here*

An `if` statement

Lets us *decide* what to do: *if True*, then *do* this.

weather.py

```
temperature = 95

if temperature > 80:
  print("It's too hot!")
  print("Stay inside!")
```

This is True

So these lines are run

```
> python3 weather.py
It's too hot!
Stay inside!
```

if Code Block

weather.py

```
temperature = 95

if temperature > 80:
  print("It's too hot!")
  print("Stay inside!")
```

Any indented code that comes after an if statement is called a code block

```
> python3 weather.py
It’s too hot!
Stay inside!
```

When the `if` statement is `False`

weather.py

```
temperature = 75

if temperature > 80:
  print("It's too hot!")
  print("Stay inside!")
```

This is False

So these lines are NOT run

```
> python3 weather.py
```

And there is no output

The Program Continues After the `if` Code Block

weather.py

```
temperature = 75

if temperature > 80:
  print("It's too hot!")
  print("Stay inside!")
print("Have a good day!")
```

This is False

The program keeps running after the if statement and its code block, so this is printed after.

```
> python3 weather.py
Have a good day!
```

Rules for Whitespace in Python

weather.py

```python
temperature = 75

if temperature > 80:
  print("It's too hot!")
    print("Stay inside!")
```

2 space indent

4 space indent

```
> python3 weather.py
File "weather.py", line 6
  print("Stay inside!")
  ^
IndentationError: unexpected indent
```

Whitespace indents in Python need to be consistent, otherwise there will be an IndentationError.

An `if`, `else` statement

weather.py

```
temperature = 75

if temperature > 80:
  print("It's too hot!")
  print("Stay inside!")
```

This is False

How do we do something else here if this is False*?*

An `if`, `else` statement

weather.py

```
temperature = 75

if temperature > 80:
  print("It's too hot!")
  print("Stay inside!")
else:
  print("Enjoy the outdoors!")
```

If this statement is False*, then run the code block below*

Otherwise, then run this code block

```
> python3 weather.py
Enjoy the outdoors!
```

if, elif, and else

weather.py

```
temperature = 50

if temperature > 80:
  print("It's too hot!")
  print("Stay inside!")
elif temperature < 60:
  print("It's too cold!")
  print("Stay inside!")
else:
  print("Enjoy the outdoors!")
```

False

True

So both of these lines are run.

```
> python3 weather.py
It's too cold!
Stay inside!
```

Can We Combine Two `if` Statements?

weather.py

```
temperature = 75

if temperature > 80:
  print("Stay inside!")
elif temperature < 60:
  print("Stay inside!")
else:
  print("Enjoy the outdoors!")
```

Let's shorten our program to only say: "Stay inside!" OR "Enjoy the outdoors!"

We're repeating `print("Stay inside!")`

Can we combine the first 2 `if` *statements?*

Logical Operator - or

weather.py

```
temperature = 75


if temperature > 80 or temperature < 60:
  print("Stay inside!")
else:
  print("Enjoy the outdoors!")
```

The keyword or *lets you combine multiple comparisons.*

At least one needs to be True *for the whole if statement to be* True

Logical Operator - or

Only *one* comparison needs to be *True* for the if statement to be *True*

weather.py

```
temperature = 75

            False    or    False  ⟶  False
if temperature > 80 or temperature < 60:
  print("Stay inside!")
else:
  print("Enjoy the outdoors!") ◀·· This is run
```

```
> python3 weather.py
Enjoy the outdoors!
```

Logical Operator - `or`

Only *one* comparison needs to be *True* for the if statement to be *True*

weather.py

```
temperature = 50
```

False or True → True

```
if temperature > 80 or temperature < 60:
  print("Stay inside!")
else:
  print("Enjoy the outdoors!")
```

This is run

```
> python3 weather.py
Stay inside!
```

Store the Forecast as a `String`

weather.py

```python
temperature = 75
forecast = "rainy"
```

Let's add another variable with the forecast as "rainy", "cloudy", or "sunny".

Logical Operator - and

Both comparisons need to be True for the if statement to be True

weather.py

```python
temperature = 75
forecast = "rainy"


if temperature < 80 and forecast != "rain":
    print("Go outside!")
else:
    print("Stay inside!")
```

Logical Operator - and

Both comparisons need to be True **for the if statement to be** True

weather.py

```
temperature = 75
forecast = "rainy"
               True    and    False ──▶ False
if temperature < 80 and forecast != "rain":
    print("Go outside!")
else:
    print("Stay inside!")   ◀·· This is run
```

Logical Operator - and

Both comparisons need to be True **for the if statement to be** True

weather.py

```python
temperature = 75
forecast = "sunny"

                True      and     True  →  True
if temperature < 80 and forecast != "rain":
    print("Go outside!")    ◀·· This is run
else:
    print("Stay inside!")
```

Logical Operator - not

The keyword not lets you negate a comparison. And can help make the statement more readable.

weather.py

```python
forecast = "rainy"



if not forecast == "rainy"
    print("Go outside!")
else:
    print("Stay inside!")
```

Logical Operator - not

weather.py

```
forecast = "rainy"

if not forecast == "rainy"
    print("Go outside!")
else:
    print("Stay inside!")
```

not True → False

Negate means make the opposite:
not True → False,
not False → True

This is run

The 3 Python Logical Operators

and

or

not

The keywords and and or let you combine multiple comparisons

The keyword not lets you negate a comparison

All of the Primitive Data Types

```
>>> amount = 10
```

int

```
>>> amount = 10.50
```

float

```
>>> name = "Sarah"
```

string

```
>>> answer = True
```

boolean

A boolean can store a True or False value

Evaluating Boolean Variables

weather.py

```
raining = True


if raining:
    print("Stay inside!")
```

You can set boolean variables to either True *or* False

This reads more like English

```
> python3 weather.py
Stay inside!
```

Evaluating Boolean Variables

weather.py

```python
raining = True
```

$\boxed{\texttt{not}}\ \boxed{\texttt{True}} \rightarrow \boxed{\texttt{False}}$

```python
if not raining:
    print("Go outside!")
else:
    print("Stay inside!")
```

This is run

```
> python3 weather.py
Stay inside!
```

A Random Rock, Paper, Scissors Game

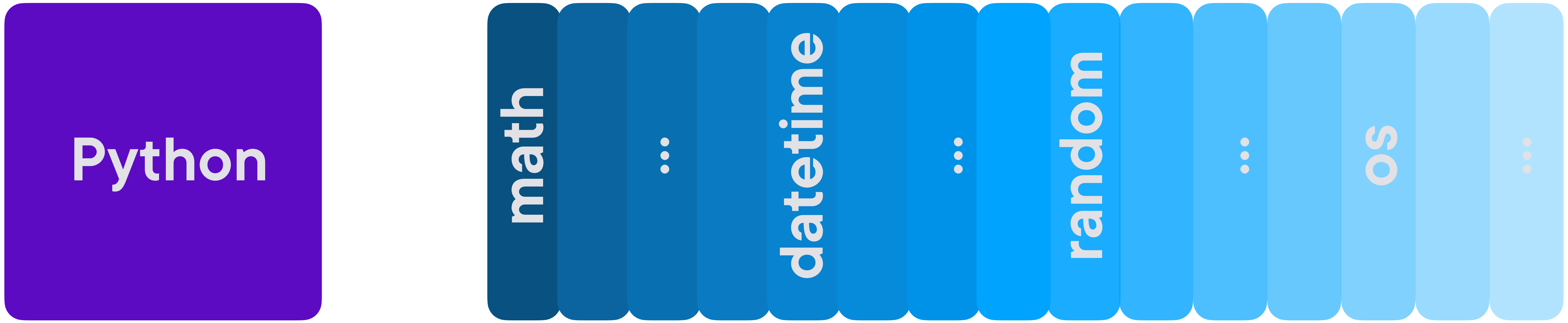
How can we randomly pick the computer's choice?

The computer has the same choice every time

```
computer_choice = 'scissors'
user_choice = input("Do you want - rock, paper, or scissors?\n")
...
```

The user gets to pick a new choice for each game

When You Install Python



Python's built-in functionality

Our programs so far have just used Python and its built-in types and functions

Python Standard Library

But if you need something extra you can import it from the Python standard library

Using the `random` Module

roll_dice.py

```
import random

roll = random.randint(1,6)
```

We need to import the module to use it

This function will return a random number between 1 and 6

Using the random Module

roll_dice.py

```
import random

roll = random.randint(1,6)

print("The computer rolled a " + str(roll))
```

Don't forget to convert the int to a string to concatenate it.

```
> python3 roll_dice.py
The computer rolled a 6
```

If we ran this more times we would see different random numbers generated.

Using the `random` Module

roll_dice.py

```python
import random

roll = random.randint(1,6)

guess = int(input('Guess the dice roll:\n'))
```

We want to convert the input to an `int` *so we can compare* `guess` *to* `roll`*.*

```
> python3 roll_dice.py
Guess the dice roll:
6
```

Using the random Module

roll_dice.py

```python
import random

roll = random.randint(1,6)

guess = int(input('Guess the dice roll:\n'))

if guess == roll:
    print("Correct! They rolled a " + str(roll))
```

```
> python3 roll_dice.py
Guess the dice roll:
6
Correct! They rolled a 6
```

Using the random Module

roll_dice.py

```python
import random

roll = random.randint(1,6)

guess = int(input('Guess the dice roll:\n'))

if guess == roll:
    print("Correct! They rolled a " + str(roll))
```

```
> python3 roll_dice.py
Guess the dice roll:
6
```

Why isn't there more output now?

We need an else statement for when the guess is wrong.

Using the random Module

roll_dice.py

```python
import random

roll = random.randint(1,6)
guess = int(input('Guess the dice roll:\n'))

if guess == roll:
    print("Correct! They rolled a " + str(roll))
else:
    print("Wrong! They rolled a " + str(roll))
```

```
> python3 roll_dice.py
Guess the dice roll:
6
Wrong! They rolled a 4
```