

Simple Python

adapted from material by Mike Scott and Bill
Young at the University of Texas at Austin

What does this code do?

```
1 print("Hello World!")  
2 print("My name is Cheese!")
```

Let's find out!

Statements

Code in Python consists of *statements*, one per line.

We can enter these lines one at a time at the interactive python prompt (sometimes called the REPL), or we can put the lines into a file and run them by typing ``python <file_name>``.

What does this code do?

```
1 print("Hello World!")
2 print("My name is Cheese!")
3 x = 7
4 print("I am")
5 print(x)
6 print("years old!")
```

What does this code do?

```
1 x = 28.5
2 y = -27
3 z = x * y - 2
4 print(z)
```

Assignment Statements

An assignment in python looks like this:

```
variable = <value>
```

We say that the variable is *assigned* **value**, or that after the assignment, **variable** "contains" **value**.

This **DOES NOT** check to see if the two values are equal, it causes an action!

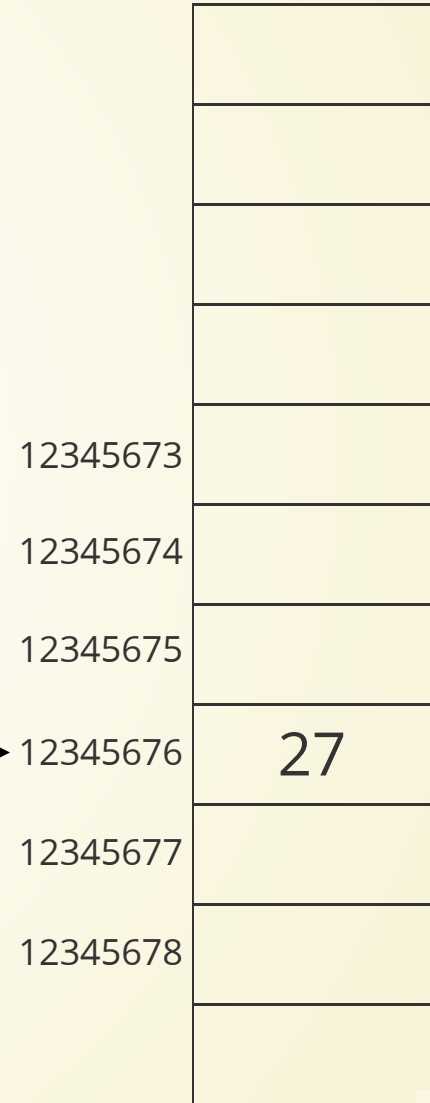
In a more sane universe, we'd use syntax like $x \leftarrow 17$, but unfortunately decisions made in the 1960s and 1970s have almost locked us into using =.



What is a variable?

- Variables on a computer are stored in *memory*.
- Memory is divided into bytes. Each byte is given an *address* or location.
- A variable is a name given to a spot in memory.
- For reasons we don't want to talk about, the rules for what memory location is chosen in Python are a little complex.

```
1 x = 27
2 # Let's assume that x is placed
3 # at memory location 12345676
```



Python

```
1 x = 3          # creates x
2 print(x)
3
4 x = "abc"      # Re-assigns x
5 print(x)
6
7 x = 3.14       # Re-assigns x again
8 print(x)
9
10 y = 6         # Creates y
11 x * y
```

You can create a new variable in Python by assigning it to a value.

Other languages require you to *declare* the variable first. You do not have to do this in Python!

Rust

```
1 let x = 7; // Okay!
2
3 y = 5;     // Not okay!
```



```
1 x = 17
```

Defines and initializes x

```
2 y = x + 3
```

Defines and initializes y

```
3 z = w
```

Error!

This code defines three variables: *x*, *y*, and *z*. On the left hand side (LHS), the variable is created if it doesn't already exist.

On the right hand side (RHS) is an expression. When the assignment statement is run, the expression on the right is *evaluated* and the result is assigned (or bound) to the variable on the left.

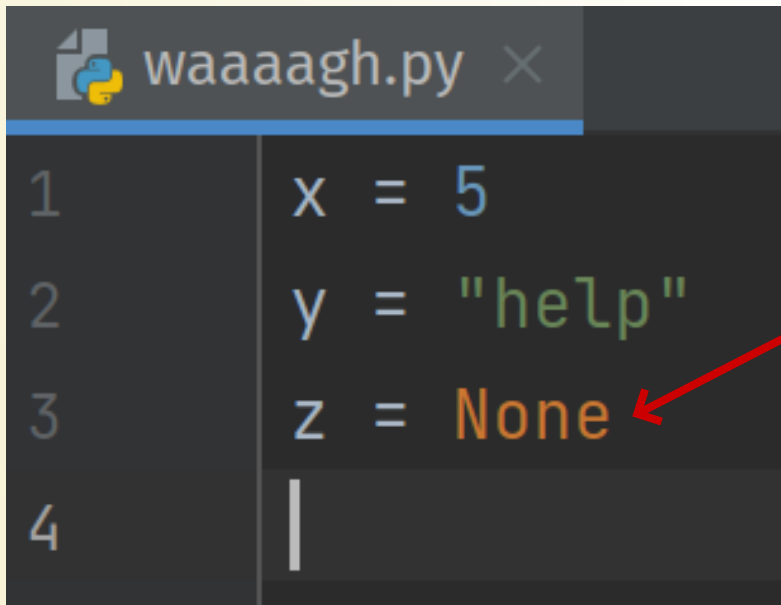
Rules About Variables Names

- Variable names have to begin with a letter or an underscore (_)
- After that, use any number of letters, underscores, or digits
- Case matters: "helpme" is different from "helpMe"
- You can't use *reserved words*

Python Reserved Words (a.k.a. Keywords)

These are words which have special meaning to Python. You cannot use them as variable names.

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



```
waaaagh.py x
1 x = 5
2 y = "help"
3 z = None
4 |
```

The screenshot shows a code editor window titled 'waaaagh.py'. The code is as follows:
1 x = 5
2 y = "help"
3 z = None
4 |
The word 'None' on line 3 is highlighted in orange. A red arrow points from the text 'Most IDEs will display these keywords in a different color to help you recognize them.' to the 'None' keyword.

Most IDEs will display these keywords in a different color to help you recognize them.

Built-ins

Python also has *built-in* functions. These are *not* reserved, but you usually don't want to override them.

```
>>> x = 12
>>> print(x)
12
>>> print = 7
>>> print(x)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'int' object is not callable
>>> █
```

```
print = 3
```

The screenshot shows a code editor with a dark background. The first line of code is `print = 3`. The word `print` is underlined with a green squiggly line, indicating a warning. A context menu is open over the code, listing the following items:

- Shadows built-in name 'print' (with a vertical ellipsis icon on the right)
- Rename the element Alt+Shift+Enter
- More actions... Alt+Enter

Below the context menu, the following code is visible:

```
mri.waaaagh  
print: int = 3
```

The code `print: int = 3` has a vertical ellipsis icon on the right side.

PyCharm will warn you if you try to do this (the green squiggle underline indicates a warning)

Python 2 Booleans

```
Python 2.7.18 (default, Oct 10 2021, 22:29:32)
[GCC 11.1.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> True = False
>>> True == False
True
>>> ???
```



Let's Play a Game!

Illegal or not?

```
1  _____ = 10
2  _123 = 11
3  ab_cd = 12
4  ab|cd = 13
5  assert = 14
6  maxValue = 100
7  print = 8
```

1. Legal (but kinda weird)
2. Legal (also kinda weird)
3. Fine
4. Illegal character
5. assert is reserved
6. Great
7. legal, but a bad idea

```
1 num_potatoes = 1000
2 weight_per_potato = 0.3
3 total_weight = num_potatoes * weight_per_potato
4 truck_capacity = 50
5 num_trucks_needed = total_weight / truck_capacity
6 print(max_truck_capacity)
```

Let's do an A/B Test!

```
1 aAaAaAaA = 1000
2 AaAaAaAa = 0.3
3 AAAAAAAAA = aAaAaAaA * AaAaAaAa
4 BBBB BBBB = 50
5 bbbbbb = AAAAAAAAA / BBBB BBBB
6 print(bbbbbb)
```


Naming Variables

- Variable names should begin with a lowercase letter
- Choose meaningful names that describe how the variable is used.
 - Use `num_columns` instead of `n`
- Use underscores to separate multiple words
- Use short names for loop variables

Types



How many ducks are in this picture?

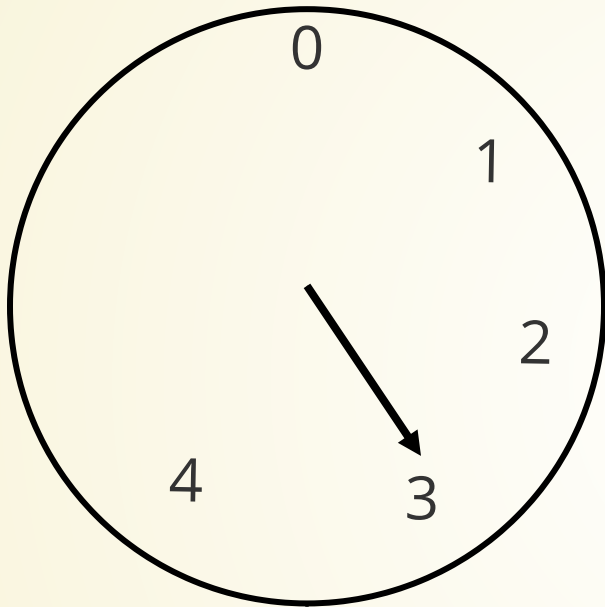
How many ducks are in this picture?



Answer: Three.

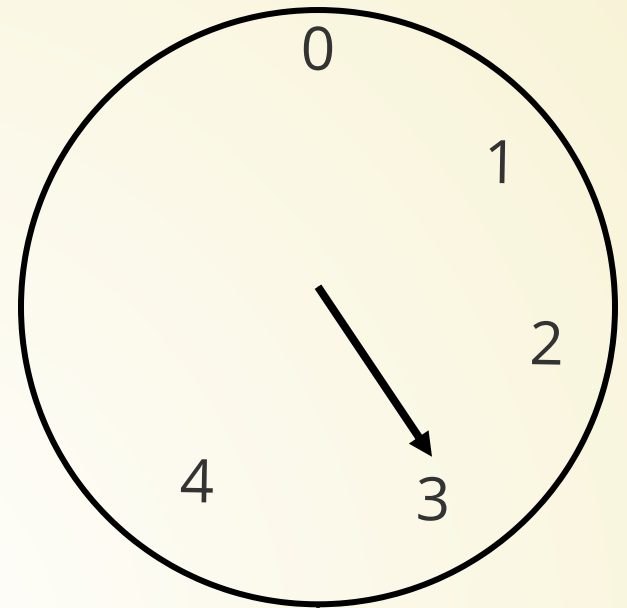
Followup question that demonstrates that I might have a screw loose: are you sure it isn't 2.8?

How much do the onions weigh?



Three pounds? Maybe it's 2.8?





Why is "maybe it's 2.8 ducks?" a weird question, but "maybe it's 2.8 pounds?" a normal question?

Data Types

A **data type** is a way to categorize values.

Whenever we assign values to variables, we also implicitly assign a type.

```
1 x = 3           # creates x and assigns an int
2 print(x)
3
4 x = "abc"      # Re-assigns x, changes to string
5 print(x)
6
7 x = 3.14       # Re-assigns x again, changes to float
8 print(x)
9
10 y = 6         # Creates y, assigns an int
11 x * y
```

Data Type	Description	Example
int	Integer. A counting number of unlimited value.	42
float	A real number.	3.1415926
str	A sequence of characters. Usually used to model text.	"Hello world"
bool	A truth value (True or False)	True, False
tuple	Immutable sequence of mixed types	(4.0, True)
list	Mutable sequence of mixed types	[1,2,3,4,5]
set	Collection without duplicates	
dict	A map from keys onto values	{'a': 3, 'b': 4}

Lots more that we won't see: complex, bytes, frozenset


```
>>> x = 17
>>> type(x)
<class 'int'>
>>> y = -20.9
>>> type(y)
<class 'float'>
>>> type(w)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'w' is not defined
>>> lst = [1,2,3]
>>> type(lst)
<class 'list'>
>>> type(20)
<class 'int'>
>>> type((2, 2,3))
<class 'tuple'>
>>> type("abc")
<class 'str'>
>>> type(print)
<class 'builtin_function_or_method'>
```

'class' is another name
for data type

The data type is a
categorization: what kind
of a thing is the value that
this variable refers to?

Other Languages

Other languages require you to state the type of a variable the first time you mention it, and the type can't change afterwards.

```
1 int main(){
2     float x = 5.0;    // x is forever a float
3     x = 7;           // Okay! 7 is converted to 7.0
4     x = "whee";      // Illegal! Trying to change type of x
5 }
```

C++

This sometimes leads people to say that Python "doesn't have types". But that isn't really true.

Three Common Types

int

signed integers (whole numbers)

- Computations are exact and of unlimited size
- Examples: 4, -17, 0, 1000000000000000000

float

signed real numbers (with decimal points)

- Large range, but fixed. Computations are *approximate*
- Examples: 3.2, -9.0, 3e23

str

represents text

- We use this for input and output
- Examples: "Hello world!"

All these types are *immutable*.

Mutability?

```
>>> x = 37
```

```
>>> x
```

```
37
```

```
>>> x = x + 10
```

```
>>> x
```

```
47
```

Didn't we just change x?

```
>>> x = 37
>>> x
37
>>> id(x)
140509108241776
>>> x = x + 10
>>> x
47
>>> id(x)
140509108242096
```

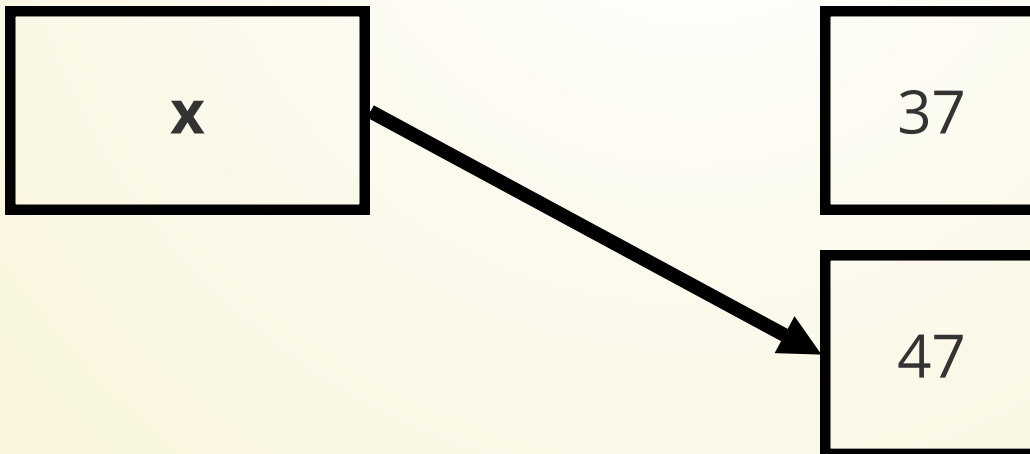
The id of x (similar to its address) have changed.

What we really did was *change* what x referred to!


$$x = 37$$



$$x = x + 10$$
$$x = 37 + 10$$
$$x = 47$$



IMMUTABLE VS MUTABLE



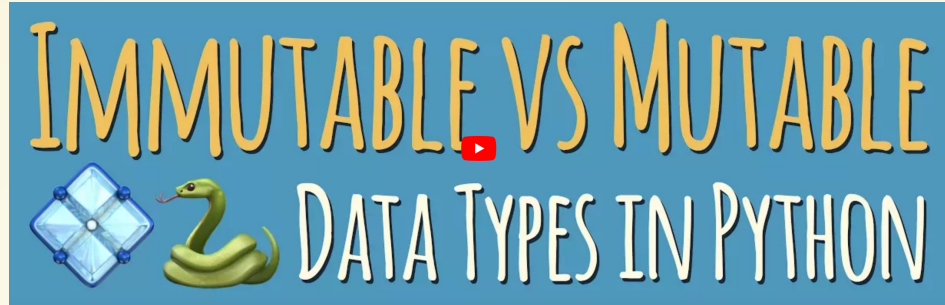
DATA TYPES IN PYTHON

An **immutable** object is one that cannot be changed by the programmer after you create it.

A **mutable** object is one that can be changed.

IMMUTABLE VS MUTABLE

DATA TYPES IN PYTHON



- An **immutable** object is one that cannot be changed by the programmer after you create it.
- This means there is usually only one copy in memory!
- When the system encounters a new reference, it creates a pointer to the already-stored value.
- There is only one 17 in memory. There is only one "abc" in memory.
- If you do something to the object that looks like you're changing it, you're actually creating a new object.

```
>>> x = 17
>>> y = 17
>>> x is y
True
>>> id(x)
140276062470896
>>> id(y)
140276062470896
>>> s1 = "abc"
>>> s2 = "ab" + "c"
>>> s1 is s2
True
>>> id(s1)
140276061685040
>>> id(s2)
140276061685040
>>> s3 = s2.upper()
>>> print(s3)
ABC
>>> id(s3)
140276061322608
```

x and y are both 17, so both point to the unique object 17 in memory.

Creates a new string
Creates a new string? (spoiler: it doesn't)
Surprise! s1 and s2 are the same string!

But since s3 is a different string than s1 or s2, it actually is a new string

How is Data Stored?

All data on a computer is stores as a series of bits (0s and 1s) in the computer's memory.

Yes, everything.

This image is 0s and 1s.

Word documents are 0s
and 1s.

This presentation is
stored as 0s and 1s.



The python programs you write are 0s and 1s. When you run them, they're converted to a *different* set of 0s and 1s.

A key problem when designing computing systems is how to represent the data as a sequence of bits.

Digital Images

Digital images can be stored in any of the following formats
(not an exhaustive list!):

- JPEG: Joint Photographic Experts Group
- PNG: Portable Network Graphics
- GIF: Graphics Interchange Format
- TIFF: Tagged Image File
- PDF: Portable Document Format
- EPS: Encapsulated Postscript
- HEIF: High Efficiently Image Format

Each format has its own rules for how sequences of 0s and 1s
should be interpreted as images.

Fortunately for us, **we usually don't have to know how the
data is stored in memory.**

Data Storage

Memory can be thought of as a big array of bytes, where a byte is a sequence of 8 bits. Each byte has an **address** and **contents**.

729484	01001000
729485	01000101
729486	01001100
729487	01001100
729488	01001111
729489	00100000
	...

ASCII encoding for 'H'

ASCII encoding for 'E'

ASCII encoding for 'L'

ASCII encoding for 'L'

ASCII encoding for 'O'

Example: ASCII Text

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

The standard way of encoding English text in memory is ASCII. It defines 128 characters using a numeric equivalent for each character.

Some of the characters are non-printable.

Characters or small numbers can be stored in one byte. If data can't be stored in a single byte, it has to be split across a number of adjacent bytes.

The way data is encoded in bytes varies. It can depend on:

- The data type being used
- The operating system being used
- The specific type of computer being used
- Settings that are set in the computer when it is first powering on

This is **messy!** Fortunately, we don't need to worry about it most of the time! Python will take care of most of it for us.

But we do need to know that these data types are stored differently!

It would be nice to look at the string "25" and do math with it.

But the number 25 (the integer number) is represented by 00011001.

And the string "25" is represented as 00110010 00110101.

And the number 25.0 (the floating point number) is represented as 01000001
11001000 0000000 0000000000

So we can't do math on "25" directly.

Fortunately, Python gives us functions that let us convert between the forms!

```
>>> float(17)
17.0
>>> str(17)
'17'
>>> int(17.75)
17
>>> str(17.75)
'17.75'
>>> int("17")
17
>>> float("17")
17.0
>>> int(17.1)
17
>>> int(16.9)
16
>>> round(16.9)
17
>>> round(17.5)
18
```

Arithmetic

Name	Meaning	Example	Result
+	Addition	$34 + 1$	35
-	Subtraction	$34.0 - 0.1$	33.9
*	Multiplication	$300 * 30$	9000
/	Float Division	$1 / 2$	0.5
//	Floor Division	$1 // 2$	0
**	Exponentiation	$4 ** 0.5$	2.0
%	Remainder	$20 \% 3$	2

This last operation is often referred to as "x mod y"

Integer Division

Floor division specified with // operator.

This goes to the floor on a number line, and discards the remainder.

```
>>> 37 // 10
3
>>> 17 // 20
0
>>> 2.5 // 2.0
1.0
>>> -22 // 7
-4
>>> -22 // -7
3
```

Modulo Operator

$x \% y$ evaluates to the remainder of $x // y$

A handwritten long division problem on a yellow background. The divisor is 32, the dividend is 487, and the quotient is 15 with a remainder of 7. Arrows point from labels to the corresponding parts of the calculation.

Quotient	→	015
Divisor	→	32
Dividend	→	487
		0
		48
		32
		167
		160
Remainder	→	7

**Enough Talking. Let's
write a program!**

BMI Calculator

Body Mass Index is a quick calculation based on height and weight used by medical professionals to broadly categorize people.

$$\text{BMI} = \frac{\text{mass}_{kg}}{\text{height}_m^2} = \frac{\text{mass}_{lb}}{\text{height}_{in}^2} \times 703$$

Let's write a program to calculate BMI for a given height and mass.

Getting Input

To get information from the user, call built-in function `input()`

Just like we can send information to print, we can get information from `input()`. Input can take an argument which will be something that is displayed to the user.

Let's try modifying our program to use this new function!

Errors

There are three types of errors in Python:

- Syntax Error
- Runtime Error
- Logic Error

The language will catch the first two for you and (usually) print a message.

Augmented Assignment

Python, like C, C++, Java, and many other languages, provides shorthand syntax for some common assignments.

Shorthand Form	Equivalent Code
<code>i += j</code>	<code>i = i + j</code>
<code>i -= j</code>	<code>i = i - j</code>
<code>i *= j</code>	<code>i = i * j</code>
<code>i /= j</code>	<code>i = i / j</code>
<code>i //= j</code>	<code>i = i // j</code>
<code>i %= j</code>	<code>i = i % j</code>
<code>i **= j</code>	<code>i = i ** j</code>

Mixed Type Arithmetic

Most arithmetic operations behave as you would expect when mixing numeric data types.

```
>>> 5 * 3 - 4 * 6
-9
>>> 5 // 2 + 4
6
>>> 5 / 2 + 4
6.5
>>> 4.2 * 3 - 1.2
11.40000000000000000002
```

Special Assignment

Simultaneous assignments

$$m, n = 2, 3$$

means

$$m = 2$$
$$n = 3$$

Except both assignments happen at the same time

Multiple assignments

$$a = b = c = 7$$

means

$$c = 7$$
$$b = c$$
$$a = b$$

Note these happen right-to-left

What does this do?

```
x, y = y, x
```