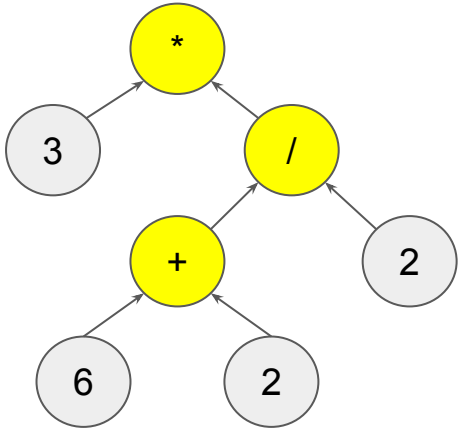
4. Operators

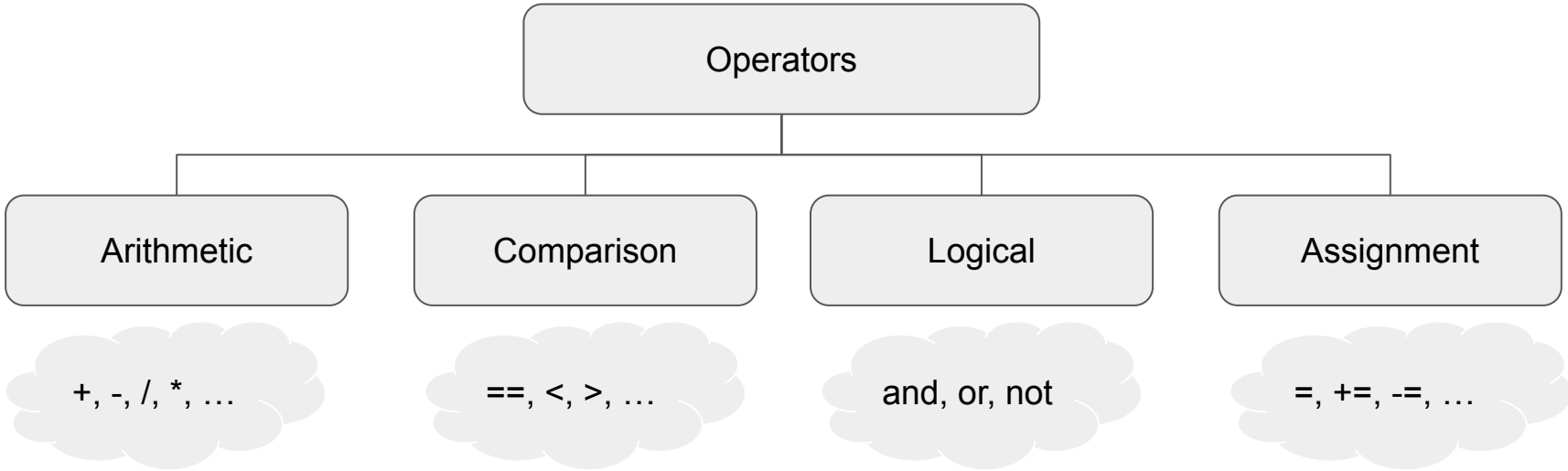
Modifying Data



What are operators?

Operators are used to **perform operations** on variables and values.

Python supports **various types** of operations, including arithmetic, comparison, logical, and assignment operations.



Arithmetic Operators

Addition	+
Subtraction	-
Multiplication	*
Division	/
Exponentiation	**
Floor division	//
Modulo	%

Arithmetic Operators

Example:

```
x = 10
y = 5
print(x + y)
print(x - y)
print(x * y)
print(x / y)
print(x ** y)
```

Arithmetic Operators

Example:

```
x = 10
y = 5
print(x + y)      -> 15
print(x - y)      -> 5
print(x * y)      -> 50
print(x / y)      -> 2.0
print(x ** y)     -> 100000
```

Comparison Operators

equal to	==
not equal to	!=
greater than	>
less than	<
greater than or equal to	>=
less than or equal to	<=

Comparison Operators

Example:

```
x = 10
y = 5
print(x == y)
print(x != y)
print(x > y)
print(x < y)
print(x >= y)
print(x <= y)
```

Comparison Operators

Example:

```
x = 10
y = 5
print(x == y)   -> False
print(x != y)   -> True
print(x > y)    -> True
print(x < y)    -> False
print(x >= y)   -> True
print(x <= y)   -> False
```

Logical Operations

`and`	Returns True if **both** the operands are True
`or`	Returns True if **either** of the operands is True
`not`	Returns True if the operand is False

X	Y	X and Y	X or Y	not(X)	not(Y)
T	T	T	T	F	F
T	F	F	T	F	T
F	T	F	T	T	F
F	F	F	F	T	T

Logical Operators

Example:

```
x = True

y = False

print(x and y)

print(x or y)

print(not x)
```

Logical Operators

Example:

```
x = True

y = False

print(x and y)   -> False

print(x or y)    -> True

print(not x)     -> False
```

Assignment Operators

Operator	Example	Same As
=	`x = 1`	`x = 1`
+=	`x += 1`	`x = x + 1`
-=	`x -= 1`	`x = x - 1`
*=	`x *= 1`	`x = x * 1`
/=	`x /= 1`	`x = x / 1`
//=	`x //= 1`	`x = x // 1`
%=	`x %= 1`	`x = x % 1`
**=	`x **= 1`	`x = x ** 1`

Assignment Operators

Example: What's the value of x after every line? Discuss with your seatmate!

```
x = 10
x += 5
x -= 3
x *= 2
x /= 4
x //= 2
x %= 2
x **= 3
```

Assignment Operators

Example:

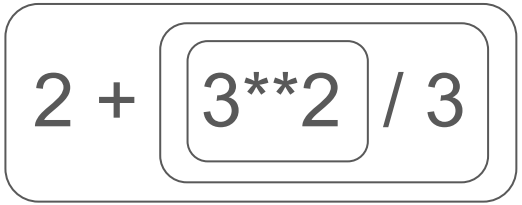
```
x = 10
x += 5      -> x = x + 5     -> x = 15
x -= 3      -> x = x - 3     -> x = 12
x *= 2      -> x = x * 2     -> x = 24
x /= 4      -> x = x / 4     -> x = 6.0
x //= 2     -> x = x // 2    -> x = 3.0
x %= 2      -> x = x % 2     -> x = 1.0
x **= 3     -> x = x ** 3    -> x = 1.0
```

NOTE

In this example, the assignments **are carried out** after each other!

Operator Precedence

Operator precedence in Python refers to the rules that **determine** the **order** in which operators are **evaluated** in an *expression*. When an expression contains multiple operators, Python follows a specific *order of precedence* to determine which operations are performed first.



Operator Precedence

Operator precedence in Python refers to the rules that **determine** the **order** in which operators are **evaluated** in an *expression*. When an expression contains multiple operators, Python follows a specific *order of precedence* to determine which operations are performed first.

2 + 3 * 4

2 + 3**2 / 3

In other words, it follows the **PEMDAS* rule**!

* Parentheses, Exponent, Multiply/Divide, Add/Subtract

Operator Precedence

`()`	Parentheses
`**`	Exponentiation
`+x  -x  ~x`	Unary plus, unary minus
`*  /  //  %`	Multiplication, division, floor division, and modulus
`+  -`	Addition and subtraction
`==  !=  >  >=  <  <=`	Comparisons, identity, and membership operators
`not`	Logical NOT
`and`	AND
`or`	OR

Operator Precedence

Example:

```
print(10 - 4 * 2)

print((10 - 4) * 2)
```

Operator Precedence

Example:

```
print(10 - 4 * 2)        → 2

print((10 - 4) * 2)      → 12
```

Associativity of Python Operators

Associativity is the order in which an expression is evaluated that has **multiple operators** of the **same precedence**. **Almost all** the operators have **left-to-right** associativity.

Example:

```
print(5 * 2 // 3)

print(5 * (2 // 3))
```

Associativity of Python Operators

Associativity is the order in which an expression is evaluated that has multiple operators of the same precedence. **Almost all** the operators have left-to-right associativity.

Example:

```
print(5 * 2 // 3)        → 3

print(5 * (2 // 3))      → 0
```

Associativity of Python Operators

Exponent operator ** has right-to-left associativity in Python.

Example:

```
print(2 ** 3 ** 2)

print((2 ** 3) ** 2)
```

Associativity of Python Operators

Exponent operator ** has right-to-left associativity in Python.

Example:

```
print(2 ** 3 ** 2)      → 512

print((2 ** 3) ** 2)    → 64
```

Associativity of Python Operators

Exponent operator ** has right-to-left associativity in Python.

Example:

```
print(2 ** 3 ** 2)      → 512
print((2 ** 3) ** 2)    → 64
```

NOTE

When in doubt, you can always use () to avoid confusion!

Other operators

EXOTIC

Identity Operators	`is, is not`
Bitwise Operators	`&, |, ^, ~, <<, >>`
List Operators (will be introduced later)	`in, not in`

String Operators

So far, we’ve been working with integers, floats, and boolean data.

Some of the operators introduced so far can also work with string data types (but not all of them).

String Operators

Name	Operator	Example
Concatenation	s1 + s2	“My name is “ + “Alice.”
Repetition	s * n	“Blah” * 3
Comparison	s1 == s2	“Alice” == “Bob”
	s1 != s2	“Alice” != “Bob”

Exercise

Define the following variables:

`x` containing an integer

`y` containing 0

`s` containing a string

Enter the following in your interpreter. What happens? Explain.

```
>>> x / y
```

```
>>> s + x
```

```
>>> s * y
```

String Formatting

As shown in the last slide, you can't concatenate a string with a number using +

Instead, an easier way is through **string formatting**!

```
name = "Alice"

age = 30

print( f"My name is {name} and I am {age} years old." )
```

String Formatting

There are various ways to format strings:

```
name = “Alice”

age = 30

print( f“My name is {name} and I am {age} years old.” )

print( “My name is {} and I am {} years old.”.format(name, age) )

print( f“My name is %s and I am %d years old.” % (name, age) )
```

String Formatting

There are various ways to format strings:

```
name = "Alice"

age = 30
```

```
print( f"My name is {name} and I am {age} years old." )

print( "My name is {} and I am {} years old.".format(name, age) )

print( f"My name is %s and I am %d years old." % (name, age) )
```

All of these do the same thing!

String Formatting: Tips and Tricks

You can limit the decimal places of a numerical value:

```
pi = 3.14159265359

formatted_pi1 = "The value of pi is {:.2f}".format(pi)

formatted_pi2 = f"The value of pi is {pi:.3f}"


print(formatted_pi1) → ???

print(formatted_pi2) → ???
```

String Formatting: Tips and Tricks

You can limit the decimal places of a numerical value:

```
pi = 3.14159265359

formatted_pi1 = "The value of pi is {:.2f}".format(pi)

formatted_pi2 = f"The value of pi is {pi:.3f}"


print(formatted_pi1) → 'The value of pi is 3.14'

print(formatted_pi2) → 'The value of pi is 3.142'
```

String Formatting: Tips and Tricks

You can limit the decimal places of a numerical value:

```
pi = 3.14159265359

formatted_pi1 = "The value of pi is {:.2f}".format(pi)

formatted_pi2 = f"The value of pi is {pi:.3f}"
```

Specifies the number of decimal places

Indicates that it should be formatted as a float

```
print(formatted_pi1) → 'The value of pi is 3.14'

print(formatted_pi2) → 'The value of pi is 3.142'
```

Escape Characters

Escape characters are special characters in strings that are prefixed with a backslash `\`

Alters the interpretation of the subsequent character in the string.

Common escape characters:

- \n Newline character
- \t Tab character
- \\ Backslash character
- \' Single quote character
- \" Double quote character

Escape Characters

```
print("Hello\nWorld!")

print("Name:\tAlice")

print("C:\\Users\\Alice\\Documents")

print('It\'s a sunny day.')

print("She said, \"Hello!\".")
```

Recap

String formatting:

```
print( f"My name is {name} and I am {age} years old." )
print( "My name is {} and I am {} years old.".format(name, age) )
print( f"My name is %s and I am %d years old." % (name, age) )
```

Common escape characters:

- \n Newline character
- \t Tab character
- \\ Backslash character
- \' Single quote character
- \" Double quote character

Exercise:

1. Open file **`ex_1.py`** (pull it from Gitlab!)
2. Define variables **`name`** and **`age`** containing a name and an age.
3. Format a string so that it prints out “My name is <NAME> and I am <AGE> years old.” (<...> are placeholders for the name and age)
4. Given variables **`length`** and **`width`**, calculate the area and perimeter of a rectangle. Print the results in a formatted string!