

Data types and operators

Big picture

- **Data**

- Information your program works with, and **types** tell the computer what kind of information it is (numbers or text). **Assignment** lets you give that kind of information a name and store it in memory so you can use it again later.

- **Operators**

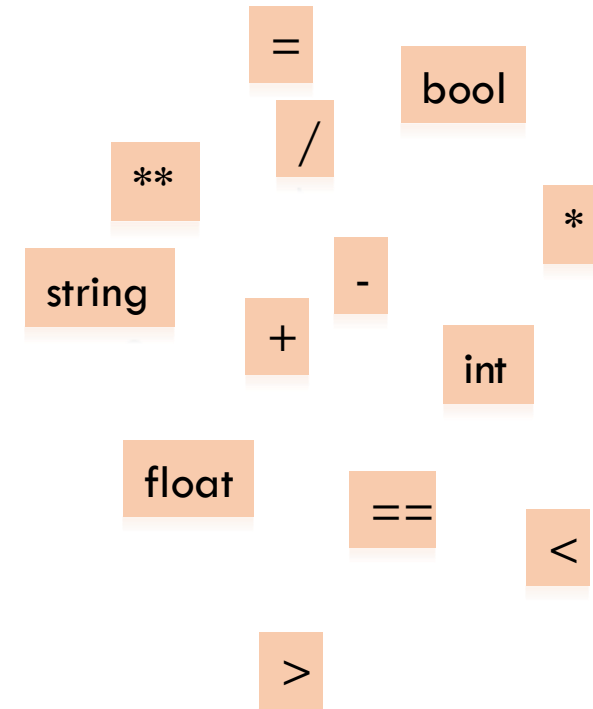
- Combine or compare data — adding, subtracting, joining, and so on.

- **Data + Operators**

- Part of the nuts, bolts, glue of programming.
- ...let you store and transform information.

Overview

- **Why types matter**
- **Kinds of data-types**
 - Strings, Floats, Integers, Booleans (type)
- **Variables, values**
 - basic data types
- **Operators**



Why types matter: bug example

- Incompatible types produce errors.

Python

```
>>> age = input("Enter age: ")
```

```
Enter age: 25
```

```
>>> age + 1
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
TypeError: can only concatenate str (not "int") to str
```

Data and types

- What kinds of data can we store into a computer?

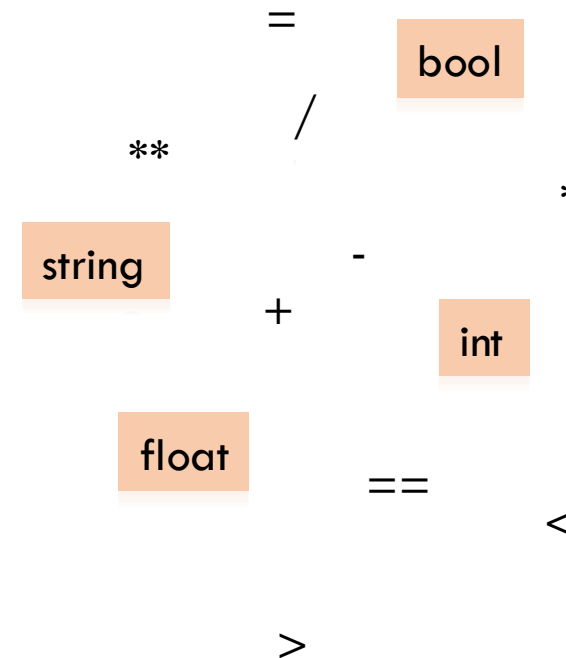
Data

- **Singular data**

- temperature, height, city name, persons age, credit card number, bank balance, area, price, weight, distance, altitude, light state, latitude, longitude, interest rate, speed, year, stadium capacity, heating state

- **Aggregate data**

- position of mouse, image, route, map, set of weights, points of interest, nutrition information, computer system information, meeting time, postcode, acceleration



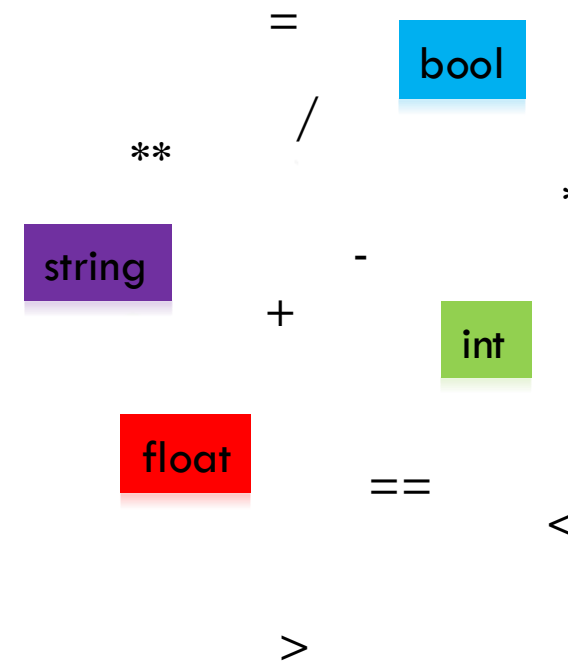
Data (classified into types)

- **Singular data**

- temperature, height, city name, persons age, credit card number, bank balance, area, price, weight, distance, altitude, light state, latitude, longitude, interest rate, speed, year, stadium capacity, heating state

- **Aggregate data**

- position of mouse, image, route, map, set of weights, points of interest, nutrition information, computer system information, meeting time, postcode, acceleration



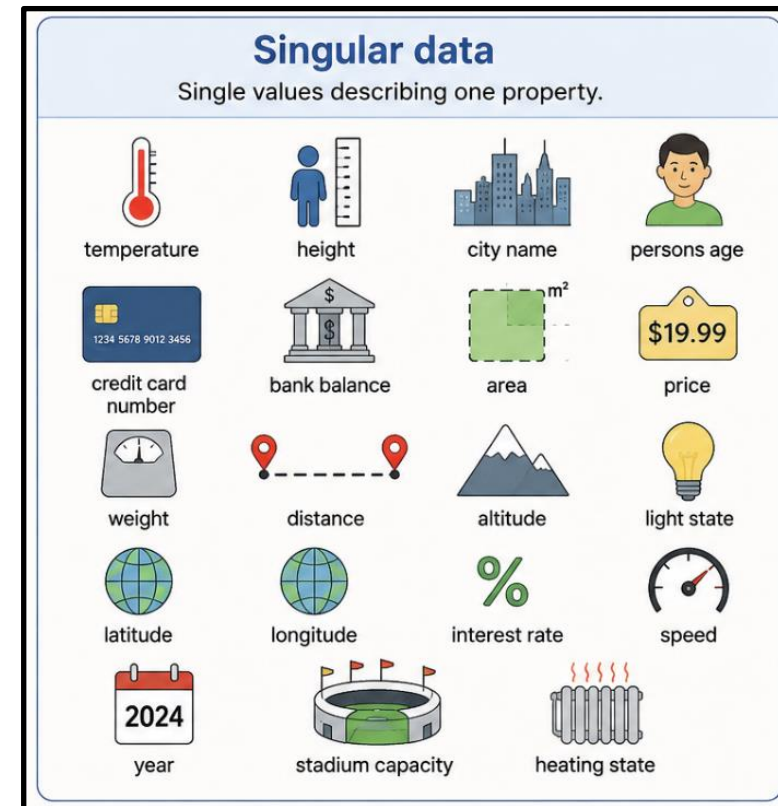
Examples (singular)

```
# --- Numbers (Integers & Floats) ---
temperature = 18.5 # float (°C)
height = 1.75 # float (meters)
persons_age = 25 # int (years)
bank_balance = 1250.50 # float
stadium_capacity = 60000 # int
year = 2026 # int

# --- Strings (Text / Identifiers) ---
city_name = "Glasgow" # str

# Credit cards are stored as strings to preserve formatting/leading zeros
# and because math is never performed on them!
credit_card_number = "4532 1234 5678 9010" # str

# --- Booleans (States / Binary values) ---
light_state = False # bool (Off)
heating_state = True # bool (On)
```



Examples (aggregate)

Python

```
# Position of mouse on screen (X, Y)
mouse_position = (1920, 1080)

# Acceleration vector across 3 axes (X, Y, Z) in m/s2
acceleration = (0.15, -9.81, 0.0)

# Latitude & Longitude coordinate pair
coordinates = (55.8642, -4.2518)
```

Python

```
# Set of physical weights recorded during a test (in kg)
set_of_weights = [70.5, 82.0, 65.3, 91.2]

# Route consisting of ordered waypoints
route = ["Glasgow", "Edinburgh", "Newcastle", "London"]
```

Aggregate data

Collections or structures containing multiple values.



position of mouse



image



route



map



set of weights



points of interest

Nutrition Facts	
Calories	120
Total Fat	3g
Carbohydrates	20g
Protein	5g

nutrition
information



computer system
information



meeting time

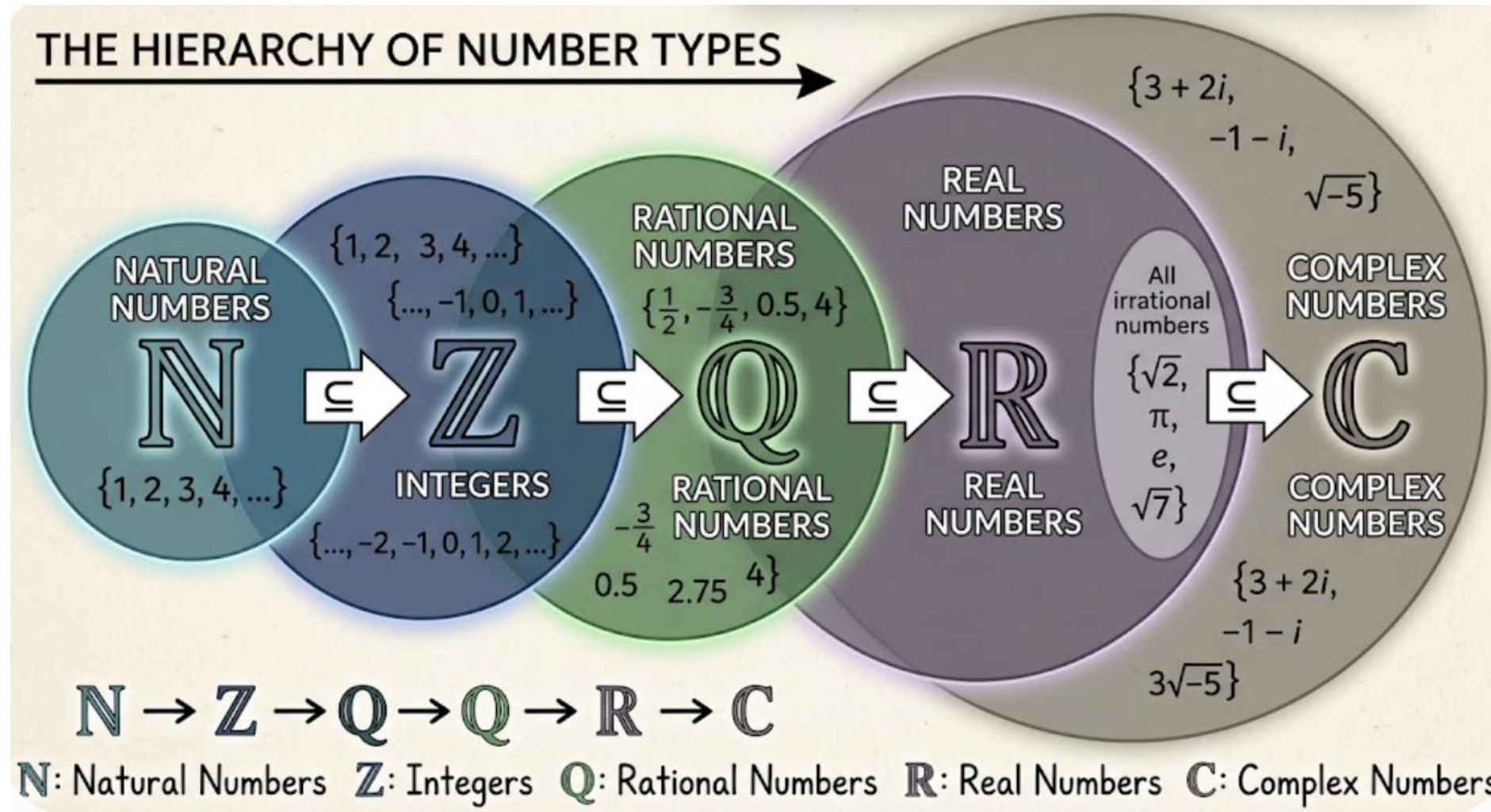


postcode



acceleration

Numbers (maths)



Numbers (Python)

- **Complex numbers supported in Python**
- **Two on this course:**

Integers 'int'

Python

```
# Basic integers
count = 42
negative_val = -100
zero = 0

# Large integers (underscores can be used for readability)
population = 8_000_000_000

# Type check
print(type(count)) # <class 'int'>
```

Real 'float'

Python

```
# Standard float
pi = 3.14159
negative_float = -0.005

# Scientific notation (3 x 10^8 = 300000000.0)
speed_of_light = 3e8

# Type check
print(type(pi)) # <class 'float'>
```

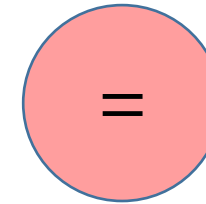
Basic operators

- Assignment: placing values into computer memory
- Operators and their context

Assignment operator, =

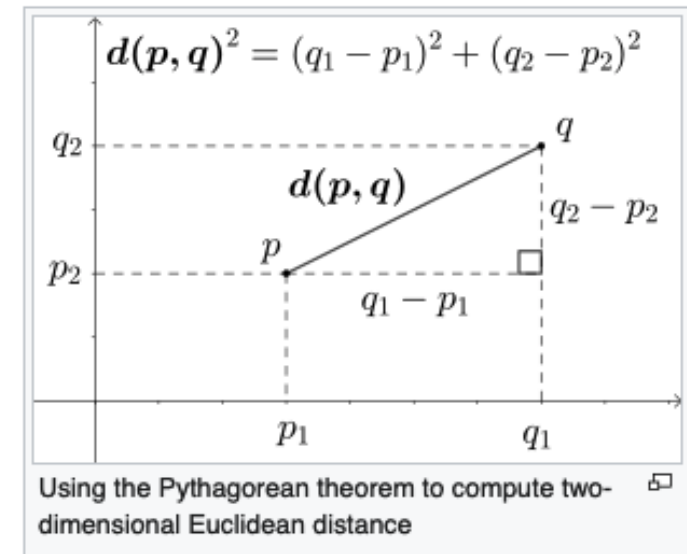
- **Most common operator**

- puts a value into memory as per stored in a named variable



- **Not an equation**

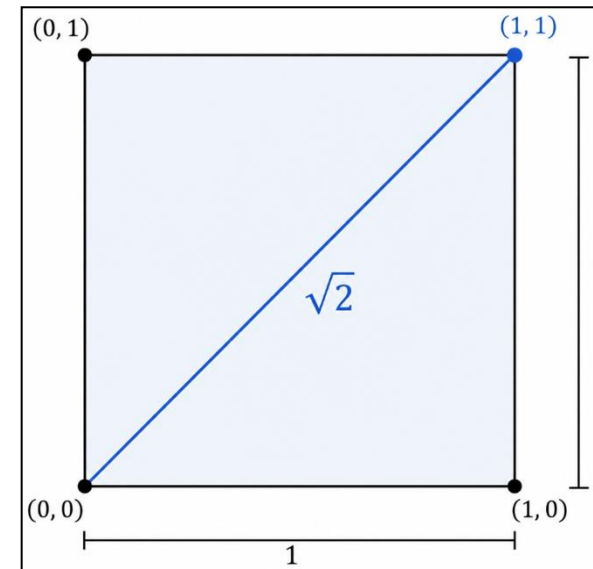
- values are equal on either side
- left-hand-side = right-hand-side



Assignment operator, =

$$D = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

```
$ python3
Python 3.14.7 (v3.14.7:823f0323ee6, Aug 5 2026, 07:07:
lang-2100.1.1.101)] on darwin
Type "help", "copyright", "credits" or "license" for mo
>>> x_1 = 0
>>> y_1 = 0
>>> x_2 = 1
>>> y_2 = 1
>>> print(x_1, y_1, x_2, y_2)
0 0 1 1
>>> distance = ((x_2 - x_1)**2 + (y_2 - y_1)**2)**0.5
>>> print(distance)
1.4142135623730951
>>> █
```



Operators

- **Basic ones** `[+, -, *, /, %]`

- + (add)

- (subtract)

- * (multiply)

- / (divide)

- % (remainder or 'mod' operator)

- **Depend on context use**

- what happened depends on the type in question

```
def operators_examples():  
    # --- Addition (+) ---  
    total = 15 + 7  
    print("15 + 7 =", total)    # 22  
  
    greeting = "Hello" + " " + "World"  
    print("String concatenation:", greeting)    # "Hello World"  
  
    # --- Subtraction (-) ---  
    balance = 100 - 35  
    print("100 - 35 =", balance)    # 65  
  
    # --- Multiplication (*) ---  
    area = 5 * 8  
    print("5 * 8 =", area)    # 40  
  
    laugh = "ha" * 3  
    print("String repetition:", laugh)    # "hahaha"  
  
    # --- Division (/) ---  
    average = 100 / 4  
    print("100 / 4 =", average)    # 25.0  
  
    # --- Division (//) ---  
    average = 100 // 4  
    print("100 // 4 =", average)    # 25
```

Operators (numeric context)

Operator	Function	Example
Addition (+)	Adds two numbers	<code>5 + 3 → 8</code>
Subtraction (-)	Subtracts one number from another	<code>10 - 4 → 6</code>
Multiplication (*)	Multiplies two numbers	<code>4 * 3 → 12</code>
Division (/)	Divides numbers and returns a float	<code>7 / 2 → 3.5</code>
Floor Division (//)	Divides and rounds down to the nearest whole integer	<code>7 // 2 → 3</code>
Modulus (%)	Returns the remainder of a division	<code>7 % 2 → 1</code>
Exponentiation (**)	Raises a number to the power of another	<code>2 ** 3 → 8</code>

Operators (string context)

Operator	Function	Example
Concatenation (<code>+</code>)	Joins two strings together	<code>"Hello " + "world"</code> → <code>"Hello world"</code>
Assignment (<code>=</code>)	Assigns text to a variable	<code>s = "hello"</code>
Augmented Assignment (<code>+=</code>)	Appends text to an existing string variable	<code>s += " world"</code> → <code>"hello world"</code>
Repetition (<code>*</code>)	Repeats a string a specified number of times	<code>"ha" * 3</code> → <code>"hahaha"</code>

Augmented assignment

- **Augmented assignment**

- Apply operation to a variable then storing the result back into that same variable.

- **Shortcut**

- Combines an arithmetic or other operator with the assignment operator =
- Instead of writing the operation and assignment separately, both are performed in a single step.

```
def assignment_examples():  
    # Start with assignment (=)  
    num = 10  
    print("Initial value (num = 10):", num)  
  
    # Add and assign (+=)  
    num += 5  
    print("After num += 5:", num)  
  
    # Subtract and assign (-=)  
    num -= 3  
    print("After num -= 3:", num)  
  
    # Multiply and assign (*=)  
    num *= 2  
    print("After num *= 2:", num)
```

Type conversion

- Changing one type into a different type
- Checking the type

General behaviour

- **an_int = int(type)**
 - convert some type to an integer
- **a_float = float(type)**
 - convert some type to an integer
- **a_str = str(type)**
 - convert some type to an integer
- **a_boolean = bool(type)**
 - convert some type to an integer
- **v_type = type(type)**
 - get information on a type
 - <class 'type'> returned

Examples

```
# str -> int
num_str = "100"
num_int = int(num_str)
print(num_int, type(num_int)) # Output: 100 <class 'int'>

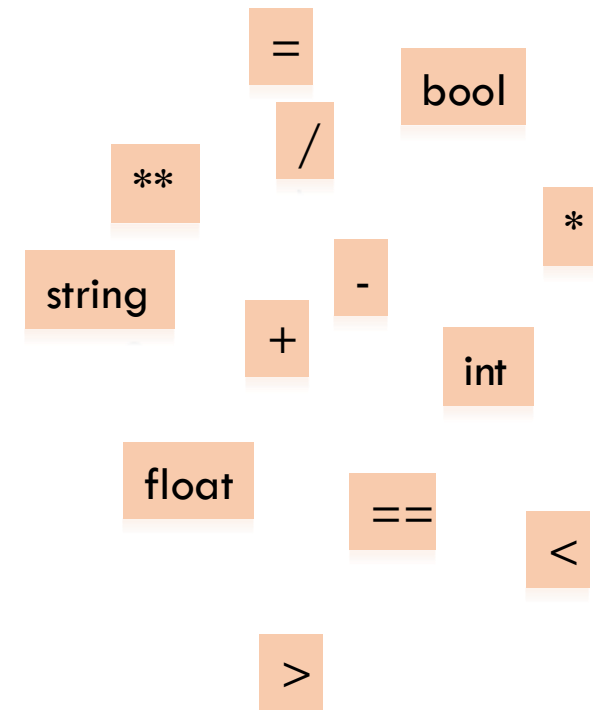
# str -> float
price_str = "19.99"
price_float = float(price_str)
print(price_float, type(price_float)) # Output: 19.99 <class 'float'>

# int/float -> str
age = 25
age_str = str(age)
print("Age: " + age_str) # Output: Age: 25

# float -> int (Truncation, not rounding)
pi = 3.99
pi_int = int(pi)
print(pi_int) # Output: 3
```

Conclusions

- **Every value in Python has a type that determines**
 - which operations make sense
 - which operations don't make sense (produce errors)
- **Getting types right prevents bugs**
 - So, be careful which types you choose for given problem
- **Variables are names pointing at values in memory**
 - `=` assigns value to memory, not mathematical equality
- **Operator behaviour depends on the type context**
 - e.g. `+` adds numbers but concatenates strings
- **Convert between types explicitly**
 - `int()`, `float()`, `str()`, `bool()`
 - check a value's type with `type()`
- **Types and operators are foundational**
 - think about what you are doing
 - master them with practice



Thanks!