EDUFY

Python data types: int, float, str, bool, list, tuple, dict and set

Maximilian Bahrami · 11 November 2026

Meet the eight types you use every day in Python: whole numbers, decimals, text, True and False, lists, tuples, dictionaries and sets. What each one is for, how to check a value's type, and how to change it. Every example runs in the page.

Every value in Python has a **type**: a whole number, a piece of text, a list of things... The type decides what you can do with the value. 10 + 5 is 15, but "10" + "5" is "105". In this first lesson you'll meet the eight types you'll use every day: int, float, str, bool, list, tuple, dict and set. You'll see what each one is for, and how to check a value's type yourself. Every example runs right here: press **Run**, then **Edit** to change it.

Variables and print(), in one minute

Two things you need first. A **variable** is a name for a value, made with =. print() shows a value. Everything after # is a **comment**, a note Python skips:

```
a = 10 # a is now 10
print(a) # shows 10
```

And type() tells you a value's type:

```
a = 10
print(type(a))
```

<class 'int'> means: this is an **int**. Now let's meet them all.

Numbers: int and float

An **int** (integer) is a whole number, with no decimal point. A **float** is a number with a decimal point:

```
a = 10 # int
b = 10.5 # float

print(a, type(a))
print(b, type(b))
print(a + b) # 20.5: an int plus a float gives a float
```

```
print(7 / 2) # 3.5: dividing always gives a float
print(7 // 2) # 3: two slashes divide and drop the decimals
```

Use an int for things you count (people, clicks, lives in a game), and a float for things you measure (prices, heights, temperatures).

Decimals are written with a dot

> Python writes 10.5, never 10,5. A comma means something else: 10,5 is two values, 10 and 5.

Text: str

A **str** (string) is text. You write it in quotes, "double" or 'single', as long as both ends match:

```
c = "Hello"
name = 'Ada'

print(c, name)
print(type(c))
print(len(c)) # 5: how many characters
print(c.upper()) # HELLO
print(c + " " + name) # joining strings with +
```

The quotes are what make it a string. 10 is a number, but "10" is text that happens to contain digits:

```
print(10 + 10) # 20
print("10" + "10") # 1010: two strings joined
```

Yes or no: bool

A **bool** (boolean) is either True or False. In Python they start with a **capital letter**:

```
h = True
is_raining = False

print(h, type(h))
print(5 > 3) # True: a comparison gives a bool
print(5 == 3) # False: == asks "is it equal?"
```

true (lower case) is an error in Python: it looks for a variable called true. You'll use bools all the time in the lesson on if.

Many values in order: list

A **list** keeps several values in order, inside **square brackets**. You can change it: add, remove and replace items.

```
d = [1, 2, 3, 4, 5]
fruits = ["apple", "banana", "cherry"]

print(fruits[0]) # apple: counting starts at 0
print(fruits[-1]) # cherry: -1 is the last one
print(len(fruits)) # 3

fruits.append("mango") # add to the end
fruits[1] = "kiwi" # replace the second item
print(fruits)
```

The position of an item is its **index**, and the first index is **0**, not 1. A list can hold any type, even mixed: [1, "two", 3.0, True].

A list that never changes: tuple

A **tuple** is like a list, but in **round brackets**, and it can't be changed after you make it. That's called **immutable**:

```
f = (1, 2, 3, 4, 5)
point = (3, 7)

print(f[0]) # 1: reading works like a list
print(point)

point[0] = 10 # changing it doesn't
```

The last line stops with a TypeError. That's the point of a tuple: use it for values that belong together and shouldn't change, like the x and y of a point, or a date (2026, 11, 11).

Names and values: dict

A **dict** (dictionary) stores **key-value pairs**, in **curly brackets**. Instead of asking "what's at position 0?", you ask "what's the value for name?":

```
e = {"name": "John", "age": 30}

print(e["name"]) # John
print(e["age"]) # 30

e["age"] = 31 # change a value
e["city"] = "Lund" # add a new pair
```

```
print(e)
print(e.keys())
```

Each key appears only once. Use a dict when your data has labels: a user (name, email, age), a product (title, price), the settings of an app.

Is a dict ordered?

> Since Python 3.7, a dict keeps the order you added the items in, so print(e) shows them in that order. But you never look up "the first item": you look up by **key**, e["name"]. Think of it as a set of labels, not a numbered list.

Unique values: set

A **set** holds values **without duplicates** and **without order**, also in curly brackets, but with no key: parts:

```
g = {"a", "b", "c", "d", "e"}
numbers = {1, 2, 2, 3, 3, 3}

print(numbers) # {1, 2, 3}: duplicates are dropped
print("a" in g) # True: is it in the set?
print(len(g)) # 5
```

Because a set has no order, it has no index: g[0] is an error, and printing it may show the items in a different order each time. Use a set to remove duplicates, or to check quickly whether something is in a group:

```
visitors = ["ana", "bo", "ana", "cy", "bo"]
unique = set(visitors)
print(unique)
print(len(unique), "different visitors")
```

All eight at a glance

Type	Example	Brackets	Ordered?	Can change?	Use it for
int	10				counting
float	10.5				measuring
str	"Hello"	quotes	yes	no	text
bool	True				yes or no
list	[1, 2, 3]	[ ]	yes	yes	things in order

Type	Example	Brackets	Ordered?	Can change?	Use it for
tuple	(1, 2, 3)	( )	yes	no	fixed groups
dict	{"age": 30}	{ } with key: value	by key	yes	labelled data
set	{"a", "b"}	{ }	no	yes	unique values

Here are your eight from the start of the lesson, with their types:

```
a = 10
b = 10.5
c = "Hello"
d = [1, 2, 3, 4, 5]
e = {"name": "John", "age": 30}
f = (1, 2, 3, 4, 5)
g = {"a", "b", "c", "d", "e"}
h = True

for value in [a, b, c, d, e, f, g, h]:
print(type(value).__name__, "→", value)
```

(The for line goes through the values one by one. You'll learn it properly in the lesson on loops.)

Changing a type

You can turn a value into another type with the type's name:

```
print(int("42") + 1) # 43: text to a whole number
print(float("2.5") * 2) # 5.0: text to a decimal
print(str(100) + "%") # 100%: number to text
print(list("abc")) # ['a', 'b', 'c']
print(set([1, 1, 2])) # {1, 2}
```

You'll need int() a lot: when a user types a number, Python gives it to you as text.

Common mistakes

Adding text and a number

Wrong

```
age = 30
print("Age: " + age)
```

Right

```
age = 30
print("Age: " + str(age))
```

+ can add two numbers or join two strings, but not one of each. Python stops with TypeError: can only concatenate str (not "int") to str. Turn the number into text with str(). (In lesson 2 you'll see an easier way: f-strings.)

Empty curly brackets make a dict, not a set

Wrong

```
tags = {}
print(type(tags))
```

Right

```
tags = set()
print(type(tags))
```

{} is an **empty dict**. For an empty set, write set().

Counting from 1

Wrong

```
colors = ["red", "green", "blue"]
print(colors[3])
```

Right

```
colors = ["red", "green", "blue"]
print(colors[2])
```

Three items have the indexes 0, 1 and 2. colors[3] gives IndexError: list index out of range. For the last item, colors[-1] always works.

Exercise

Describe a student

Make variables for one student:

1. name: their name (a str)
2. age: their age (an int)
3. height: their height in metres, like 1.72 (a float)
4. is_member: whether they're a member (a bool)
5. grades: a list of three grades

> 6. student: a dict with the keys "name" and "age", using the variables above

> Then print the type of each one, and print the student's **second** grade.

Answer

```
name = "Lina"
age = 19
height = 1.72
is_member = True
grades = [8, 9, 7]
student = {"name": name, "age": age}

print(type(name), type(age), type(height), type(is_member))
print(type(grades), type(student))
print("Second grade:", grades[1])
```

> The second grade is grades[1], because counting starts at 0. Notice that a dict's values can be variables: {"name": name} takes the value of name and stores it under the key "name".

Quick check

What is the type of x = 3.0?

- int
- ✓ float
- str

> The decimal point makes it a float, even though the decimals are 0. 3 would be an int, and "3.0" a str.

You want to store the x and y of a point, and they should never change. Which type fits best?

- list
- ✓ tuple
- set

> A tuple keeps values in order and can't be changed: perfect for (x, y). A list could be changed by accident, and a set has no order, so you couldn't tell x from y.

What does this print?

```
s = {1, 2, 2, 3}
print(len(s))
```

- 4
- ✓ 3

- An error

A set keeps each value once, so the second 2 is dropped. The set is {1, 2, 3}, with 3 values.

Key points

- int is a whole number, float a number with a decimal point, str text in quotes, bool True or False (capital letters).
- list [ ]: ordered and changeable, indexes start at 0.
- tuple ( ): ordered, but can't be changed.
- dict {key: value}: values looked up by key.
- set { }: unique values, no order, no index. An empty set is set(), because {} is an empty dict.
- type(x) tells you the type; int(), float(), str(), list() and set() change it.