- data types
- conversions
- variables
- formatting strings
- mathematical operations
- rounding

Python

data types

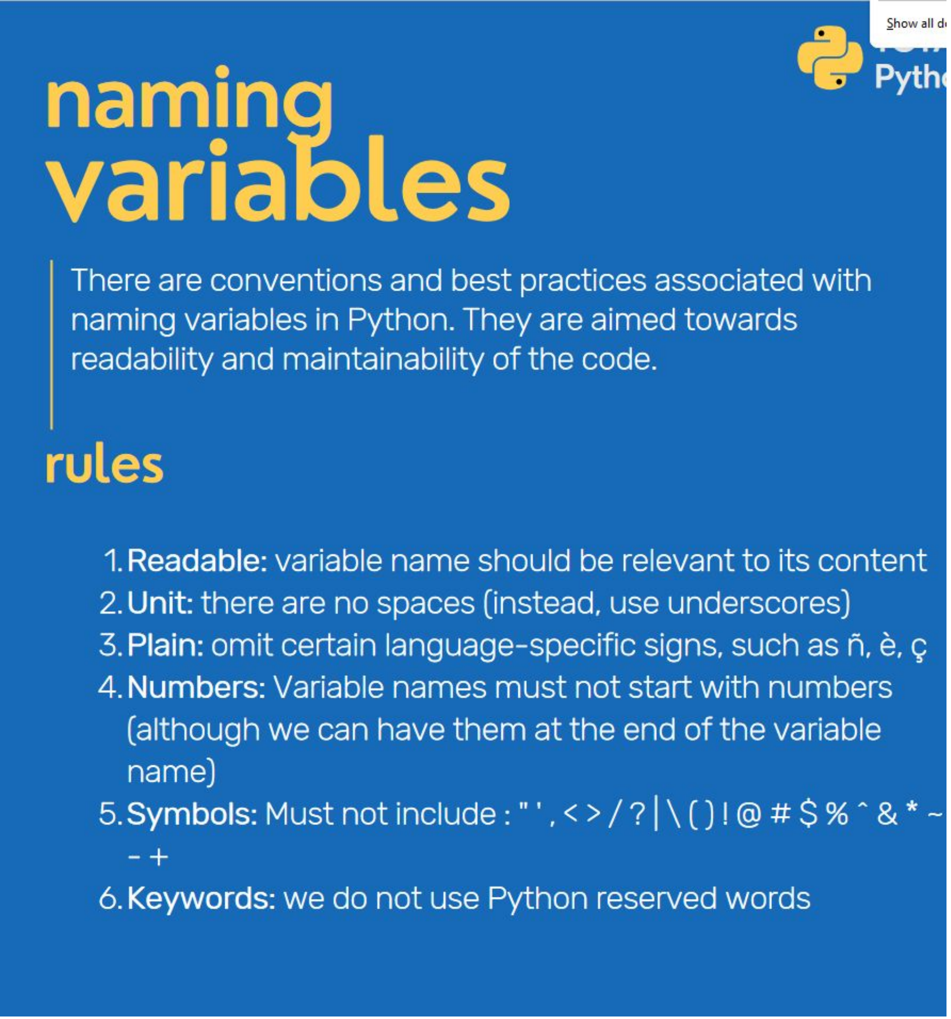
In Python we have several data structures, which are fundamental in programming since they store information and allow us to manage it.

text (str)	numbers	booleans
`"Python"`	**int** `250`	`True`
`"750"`	**float** `12.50`	`False`

structures	mutable	ordered	duplicates
lists [ ]	✓	✓	✓
tuples ( )	✗	✓	✓
sets { }	✓	✗	✗
dictionaries { }	✓	✗ *	✗ : ✓ **

*: In Python 3.7+, there are new features **: key is unique ; value can be repeated

string (str)	"hello", "%$&", " ", "123"
integer (int)	150, 1, 555, -15, 0
floating (float)	1.25, 25.0, 500.50, -95.1
lists (list)	["sea", 1, -3, 4.5, "mars", 0]
dictionaries (dict)	{'color':'blue','dog':'chop'}
tuples (tuple)	('mon','tue','wed',thu','fri')
sets (set)	{'a', 'b', 'c', 'd', 'e' }
booleanos (bool)	True, False

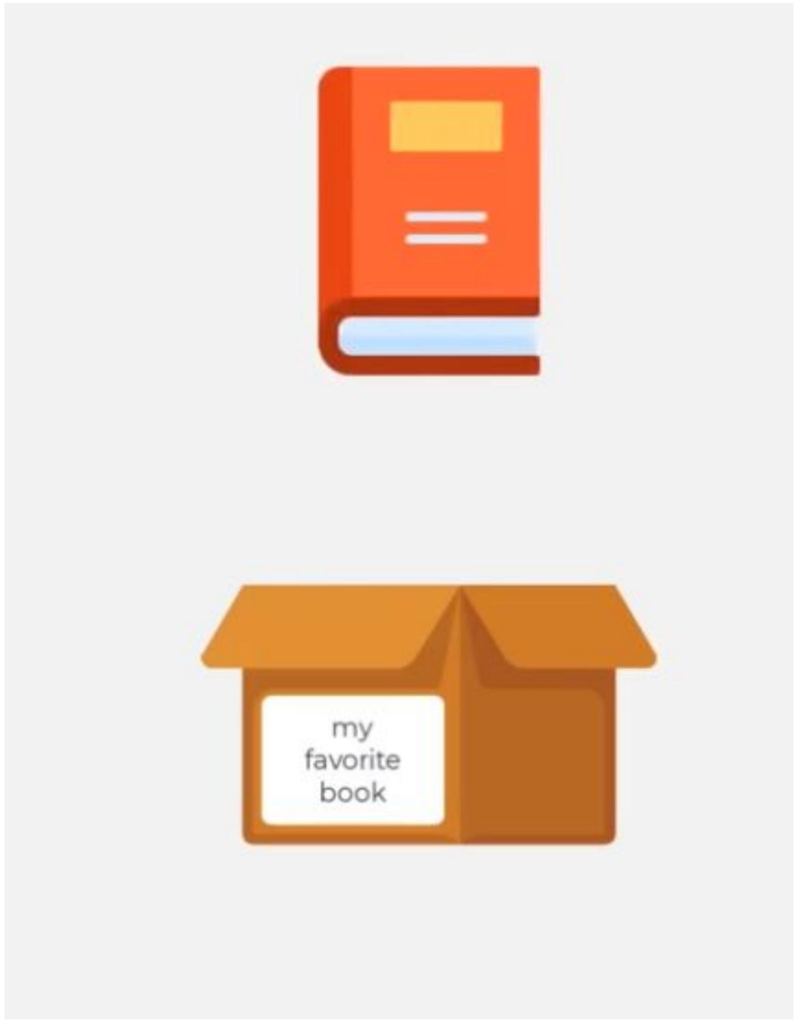

Show all d
Pyth

naming variables

There are conventions and best practices associated with naming variables in Python. They are aimed towards readability and maintainability of the code.

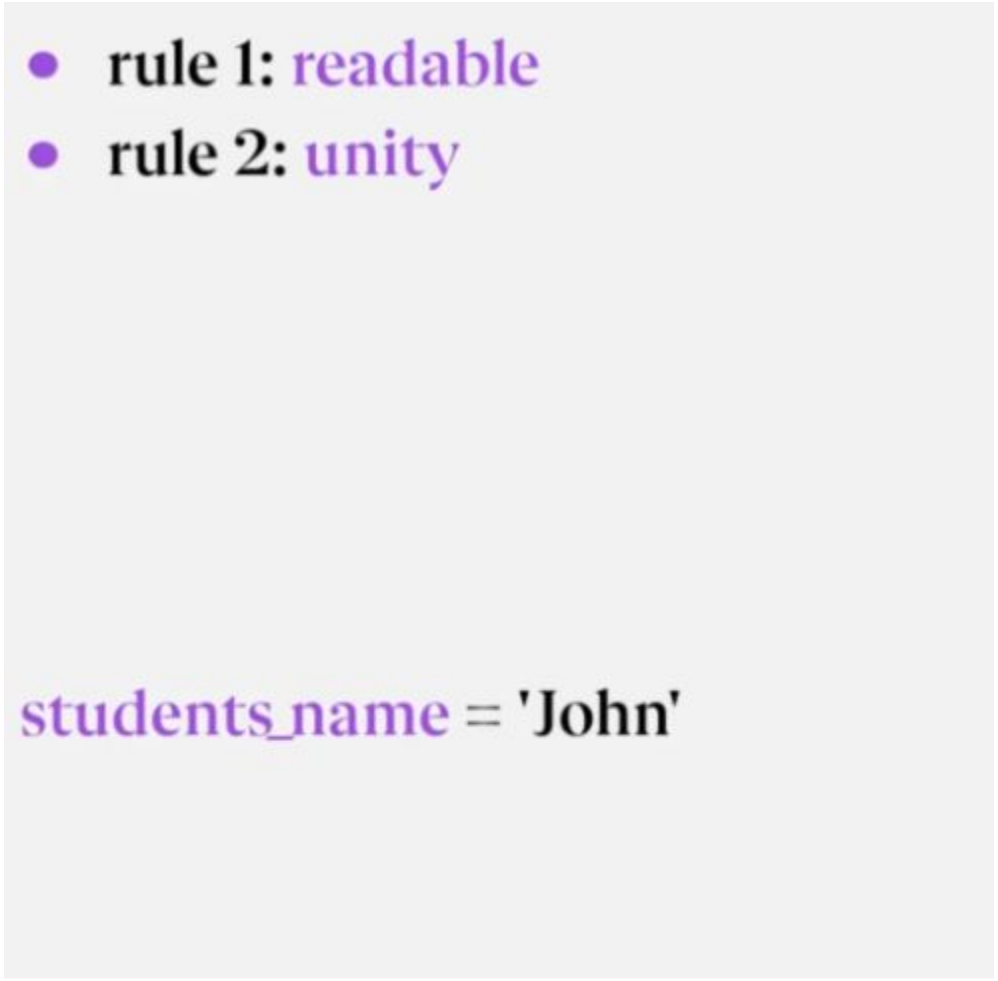
rules

1. **Readable:** variable name should be relevant to its content
2. **Unit:** there are no spaces (instead, use underscores)
3. **Plain:** omit certain language-specific signs, such as ñ, è, ç
4. **Numbers:** Variable names must not start with numbers (although we can have them at the end of the variable name)
5. **Symbols:** Must not include : " ' , < > / ? | \ ( ) ! @ # $ % ^ & * ~ - +
6. **Keywords:** we do not use Python reserved words



- **rule 1:** readable

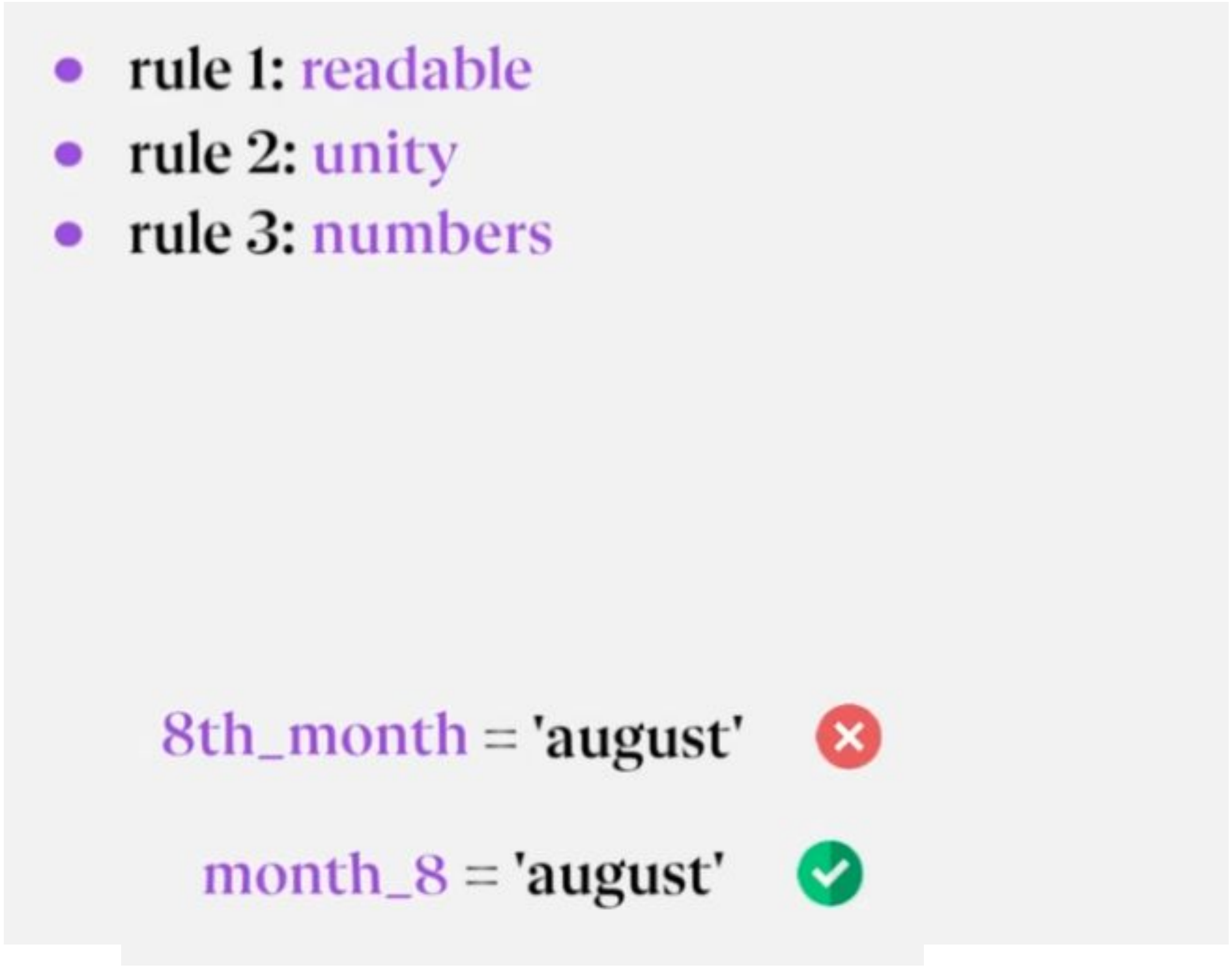
```
dog_name = 'Buster' ✓
d = 'Buster' ✓
mydoghasanameandthisisit= 'Buster' ✓
```



- rule 1: readable
- rule 2: unity

```
students_name = 'John'
```

Unity means you can’t make variables separated by spaces…

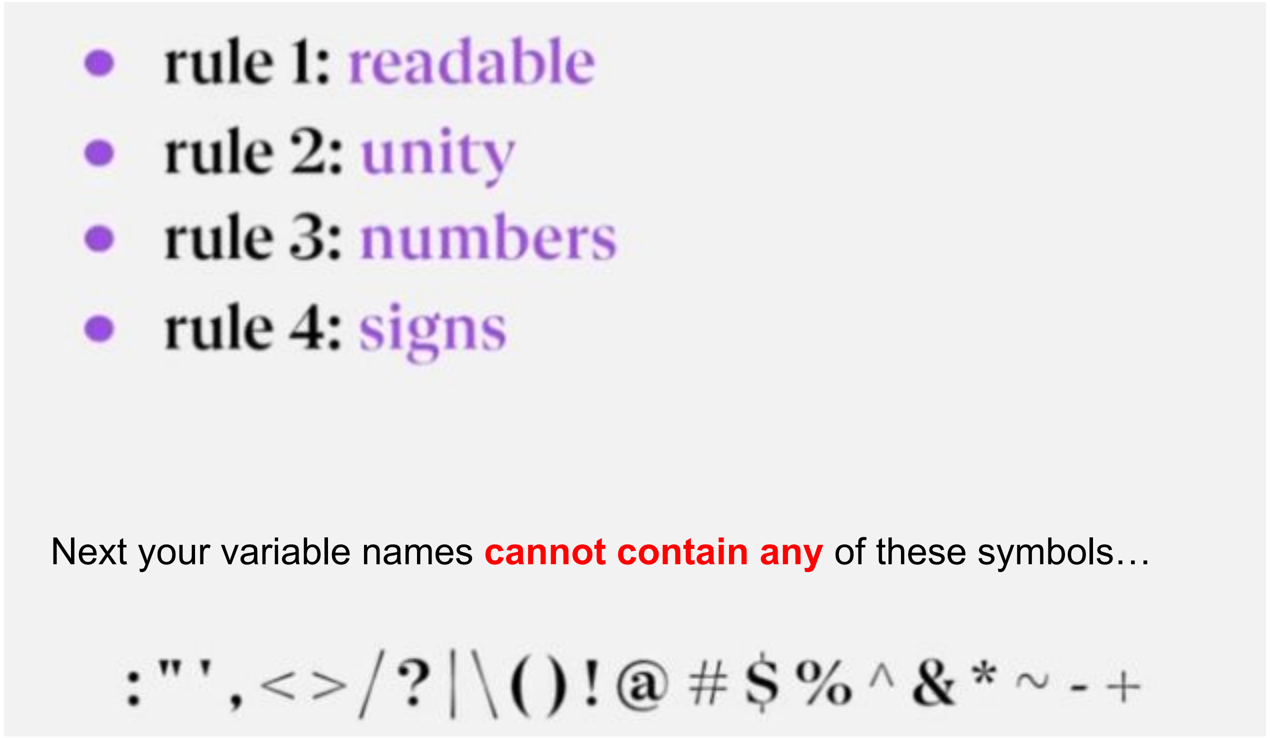


- rule 1: readable
- rule 2: unity
- rule 3: numbers

```
8th_month = 'august'   ❌

month_8 = 'august'   ✅
```

You can also use numbers in your variables just not at the beginning. ..

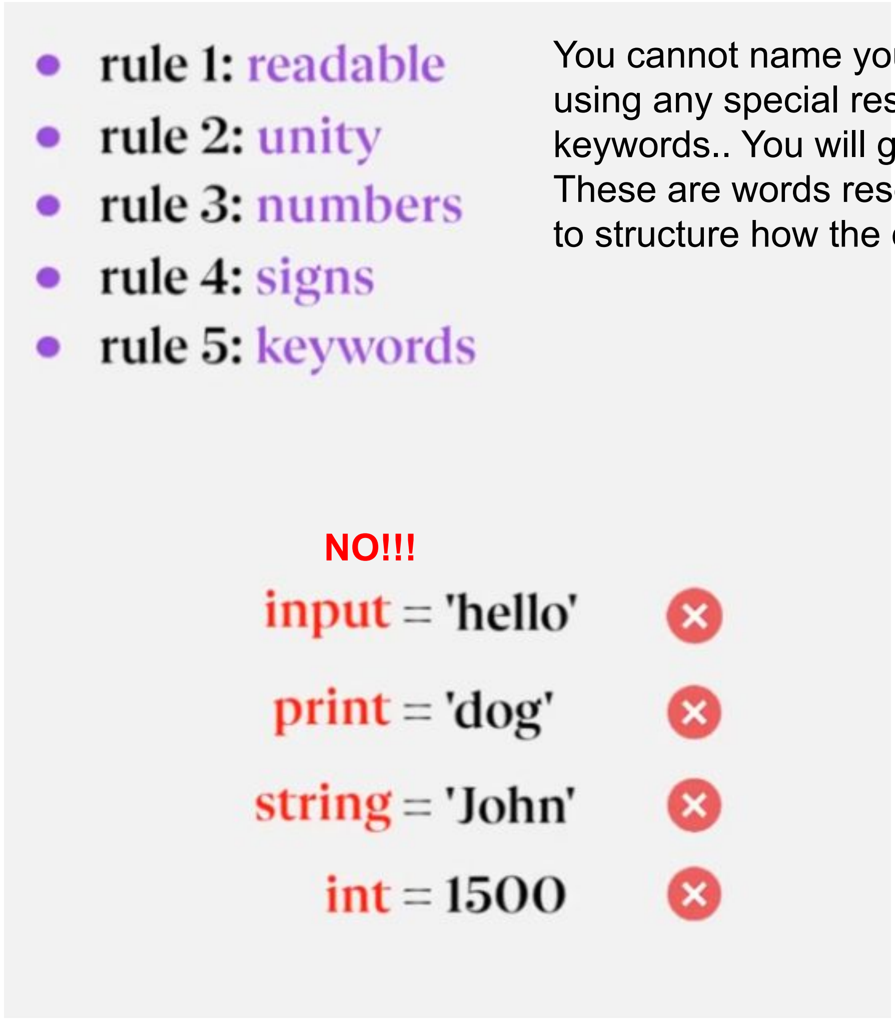
Variables must always start with a letter



- rule 1: readable
- rule 2: unity
- rule 3: numbers
- rule 4: signs

Next your variable names **cannot contain any** of these symbols…

: " ' , < > / ? | \ ( ) ! @ # $ % ^ & * ~ - +

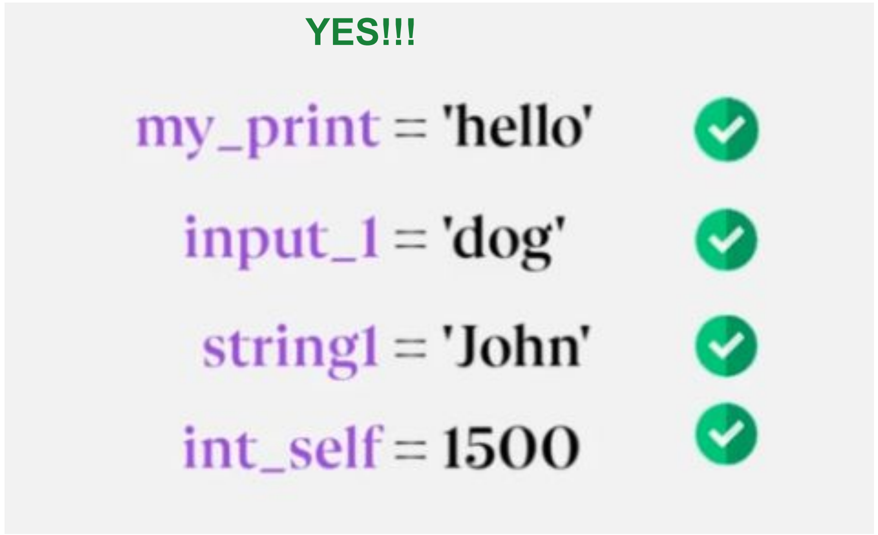


- rule 1: readable
- rule 2: unity
- rule 3: numbers
- rule 4: signs
- rule 5: keywords

You cannot name your variables using any special reserved keywords.. You will get an error. These are words reserved by python to structure how the code works…

NO!!!

```
input = 'hello'   ✗
print = 'dog'     ✗
string = 'John'   ✗
int = 1500        ✗
```

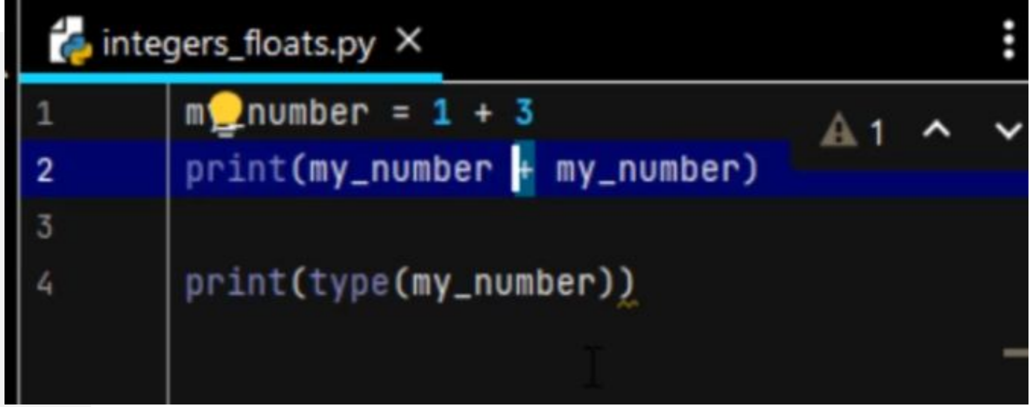


YES!!!

```
my_print = 'hello'   ✓
input_1 = 'dog'      ✓
string1 = 'John'     ✓
int_self = 1500      ✓
```

- integers

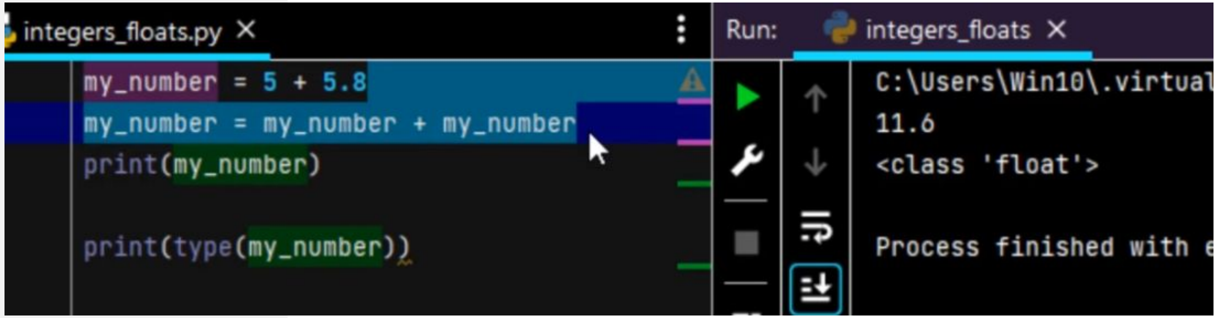
  age = 45
  population = 3.500.000
  remaining_days = 25
  rank = 563


integers_floats.py

```
my_number = 1 + 3
print(my_number + my_number)

print(type(my_number))
```

- floats

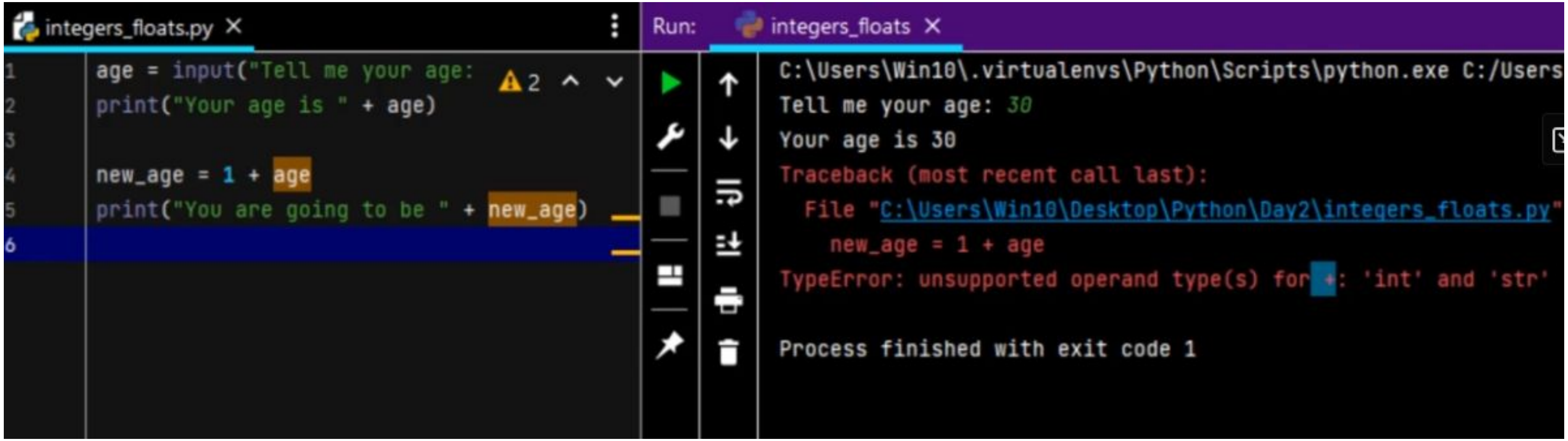
  grades = -3,5
  value = 65,71
  weight = 1,76


integers_floats.py

```
my_number = 5 + 5.8
my_number = my_number + my_number
print(my_number)

print(type(my_number))
```

Run: integers_floats

```
C:\Users\Win10\.virtual
11.6
<class 'float'>

Process finished with e
```

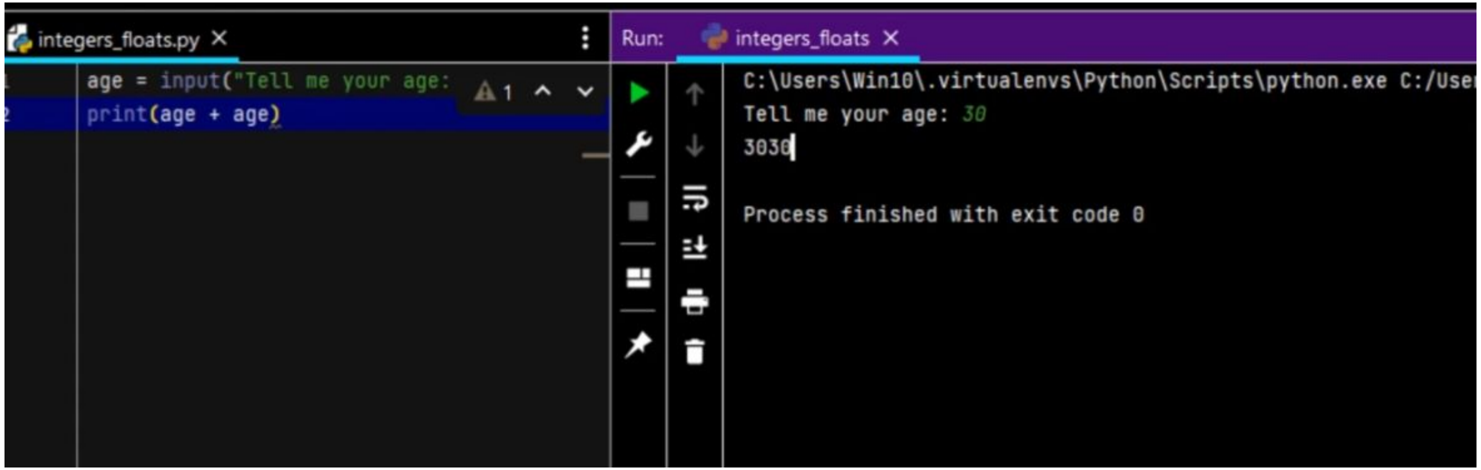


integers_floats.py

```python
age = input("Tell me your age:
print("Your age is " + age)

new_age = 1 + age
print("You are going to be " + new_age)
```

Run: integers_floats

```
C:\Users\Win10\.virtualenvs\Python\Scripts\python.exe C:/Users
Tell me your age: 30
Your age is 30
Traceback (most recent call last):
  File "C:\Users\Win10\Desktop\Python\Day2\integers_floats.py"
    new_age = 1 + age
TypeError: unsupported operand type(s) for +: 'int' and 'str'

Process finished with exit code 1
```

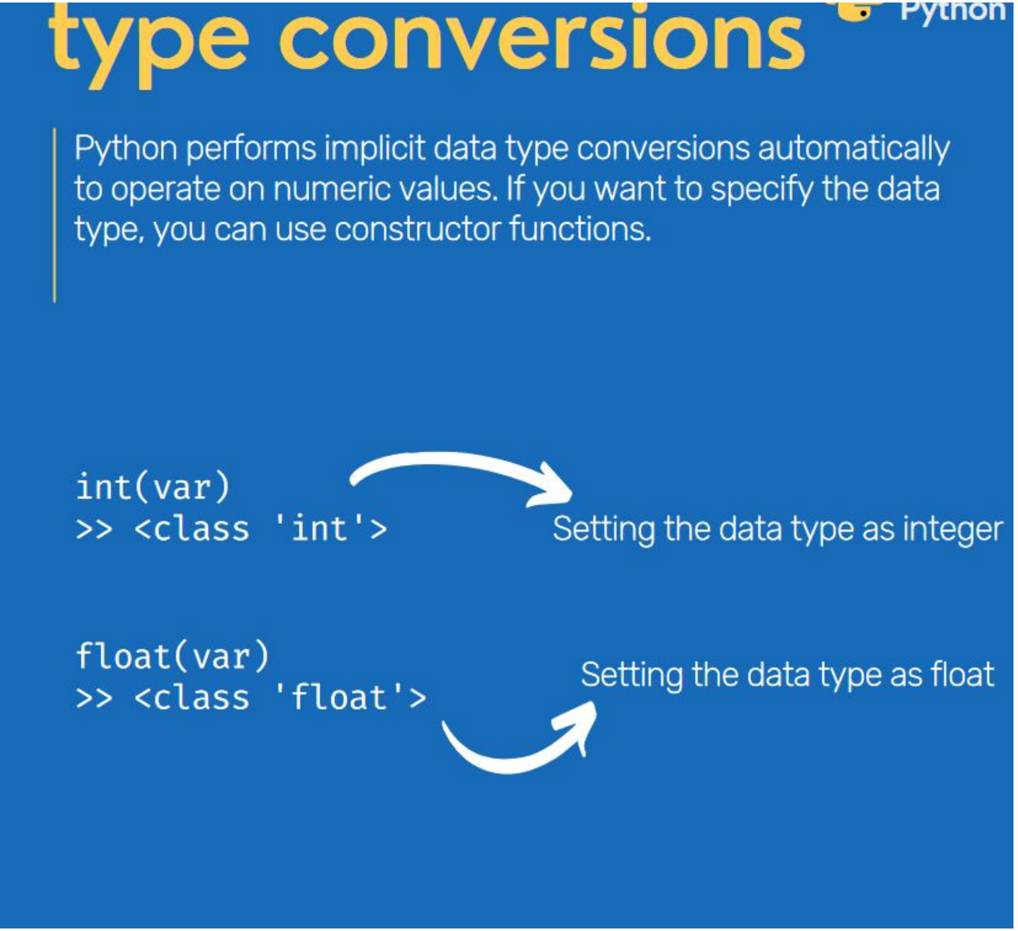


integers_floats.py

```python
age = input("Tell me your age:
print(age + age)
```

Run: integers_floats

```
C:\Users\Win10\.virtualenvs\Python\Scripts\python.exe C:/User
Tell me your age: 30
3030

Process finished with exit code 0
```



type conversions

Python

Python performs implicit data type conversions automatically to operate on numeric values. If you want to specify the data type, you can use constructor functions.

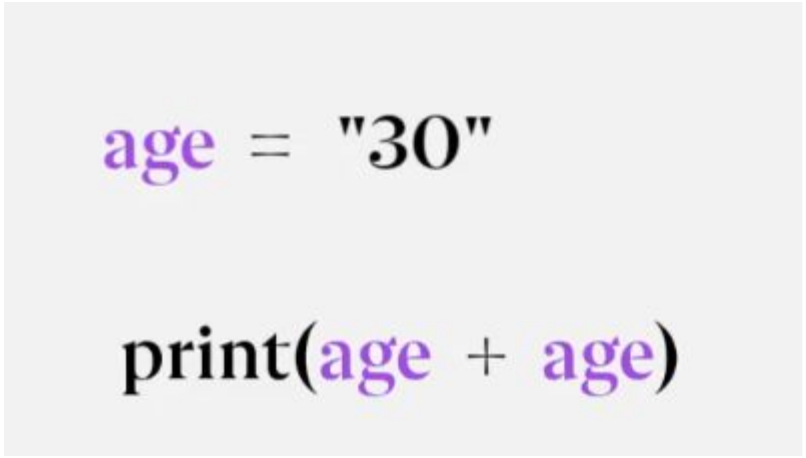
```
int(var)
>> <class 'int'>
```

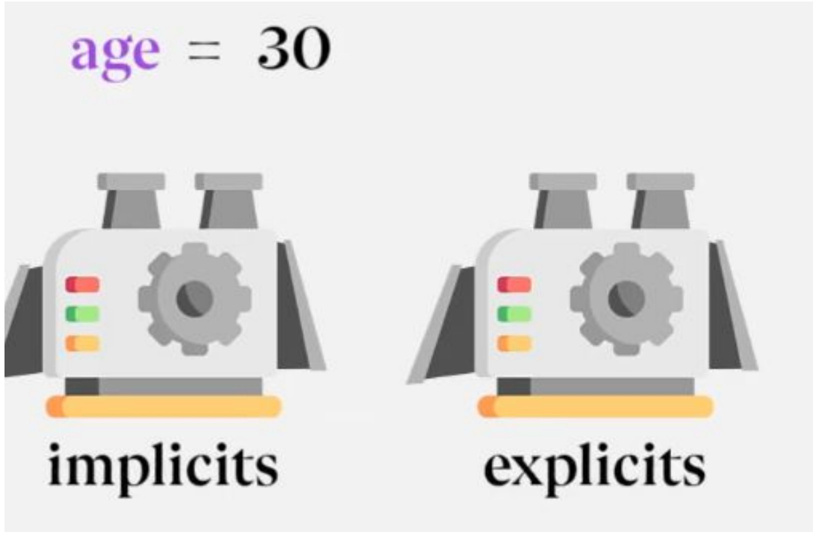
Setting the data type as integer

```
float(var)
>> <class 'float'>
```

Setting the data type as float

One of the advantages of Python about variables is that you can store any kind of data in them.
For example, the variable my data can contain an integer, then replace it with a float, then change it to a string and it would always be able to store whatever you put inside.
In most other programming languages, when you declare a variable, you have to give it a name and assign it a specific data type, for example, integer. From then on, that variable will only hold integers.
So that's what's great about variables and Python.
They only get a name and you can store any kind of data type inside them.
On the other hand, there may be situations where you need to convert the contents of a variable to a different data type.


```
age = "30"

print(age + age)
```


```
age = 30
```

when we wanted to take a user input, which is a string data type by default, and treat it as an integer in order to perform a mathematical operation on it.
The process of converting a specific data type into another is called casting, and there are two types of conversions implicit and explicit.

Implicit conversion is done automatically by Python. It happens when we perform certain operations with two different types, and Python solves it by transforming the data type without us noticing.

In contrast, explicit conversion is when we expressly ask through the code for a data type to be converted to another data type.

implicit

- Python converts the data type to another data type automatically. In this process, the user must do nothing.

explicit

- Python needs the user to do something to convert one data type to another

```
my_value = 1

another_value = string(my_value)
```

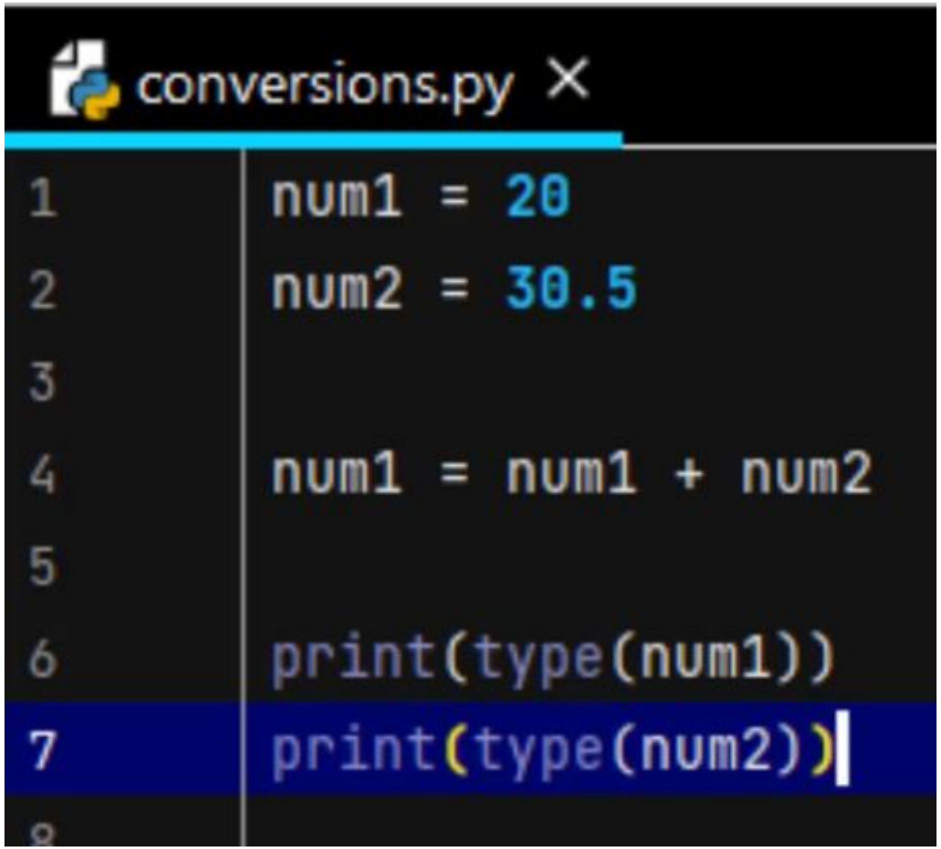
```
print(another_value)
```

For example, if I have a variable called my value that contains an integer, let's say the number one,

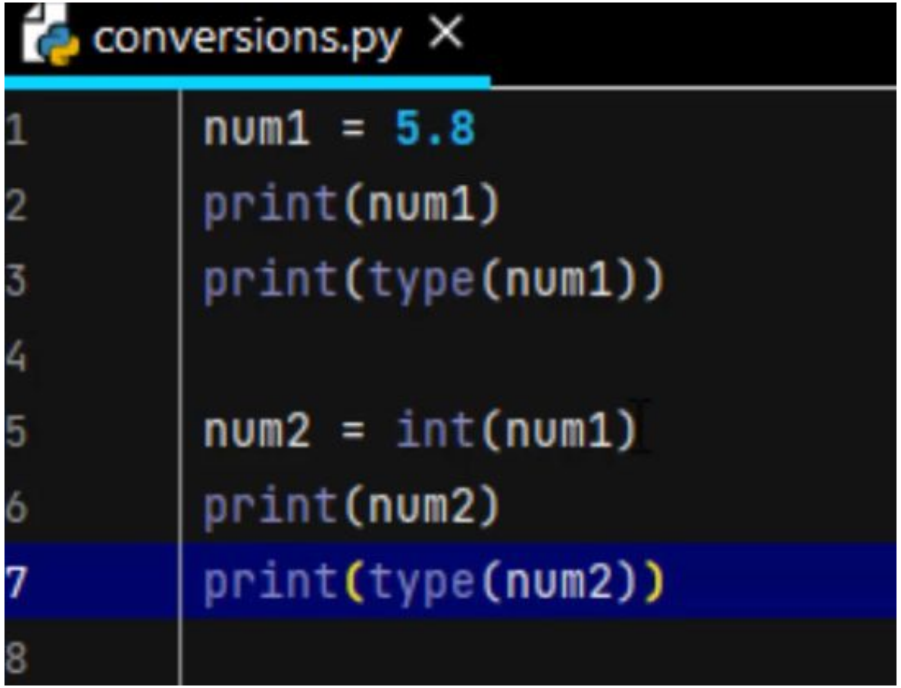
I can reassign it to another variable with a different data type, for example, a string.

Now if I print this variable, I will see the same value one.

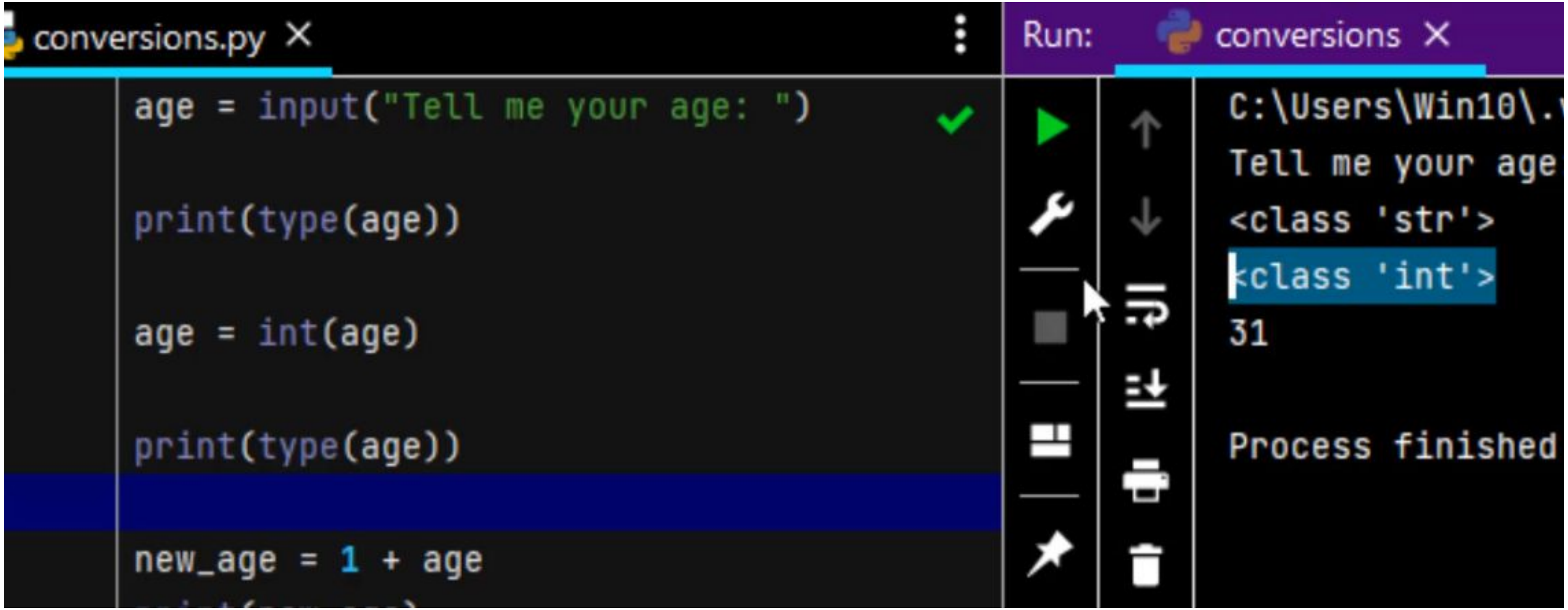
But this time is no longer an integer, it is a string.



```python
num1 = 20
num2 = 30.5

num1 = num1 + num2

print(type(num1))
print(type(num2))
```

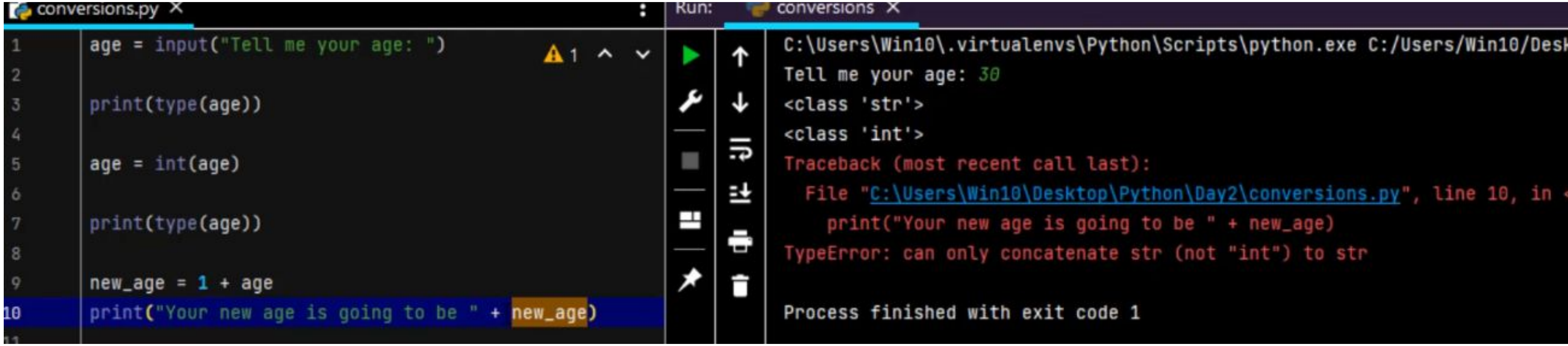


```python
num1 = 5.8
print(num1)
print(type(num1))

num2 = int(num1)
print(num2)
print(type(num2))
```



```
age = input("Tell me your age: ")

print(type(age))

age = int(age)

print(type(age))


new_age = 1 + age
```

```
C:\Users\Win10\.
Tell me your age
<class 'str'>
<class 'int'>
31

Process finished
```



conversions.py

```python
age = input("Tell me your age: ")

print(type(age))

age = int(age)

print(type(age))

new_age = 1 + age
print("Your new age is going to be " + new_age)
```

Run: conversions

```
C:\Users\Win10\.virtualenvs\Python\Scripts\python.exe C:/Users/Win10/Desk
Tell me your age: 30
<class 'str'>
<class 'int'>
Traceback (most recent call last):
  File "C:\Users\Win10\Desktop\Python\Day2\conversions.py", line 10, in <
    print("Your new age is going to be " + new_age)
TypeError: can only concatenate str (not "int") to str

Process finished with exit code 1
```

formatting strings

We have two main tools to mix static text and variables into strings:

- **Format function:** enables you to concatenate parts of a string at desired intervals. Multiple pairs of curly braces can be used while formatting the string. Python will replace the placeholders with values in order.

```
print("My car is {}, and it's license
plate is {}".format(car_color,plate))
```

- **formatted string literals (f-strings)**: the newer way to format strings (Python 3.8+), with a simple and less verbose syntax: just include f at the beggining of the string and call the variables inside curly brackets.

```
print(f"My car is {car_color} and it's
license plate is {plate}")
```

Strings Formatting Practice #1

We need to print the associate name and number within the following sentence:

```
"Dear (associate_name), your
associate number is:
(associate_number)"
```

Remember that the precision of your answer (spaces, spelling and punctuation) is very important to arrive at the correct result.

associate_name = "Jesse Pinkman"
associate_number = 399058

Dear (associate_name), your associate number is:
(associate_number)

Strings Formatting Practice #2

Tell the user the amount of points earned within the following phrase:

```
"You have earned (new_points)
points! In total, you have
accumulated (total_points)
points"
```

Remember that the precision of your answer (spaces, spelling and punctuation) is very important to arrive at the correct result.

```
total_points = 1225
```

Strings Formatting Practice #3

Tell the user the amount of points earned within the following phrase:

```
"You have earned (new_points)
points! In total, you have
accumulated (total_points)
points"
```

This time, the amount of points accumulated (`total_points`) will be equal to the `previous_points` plus the `new_points`.

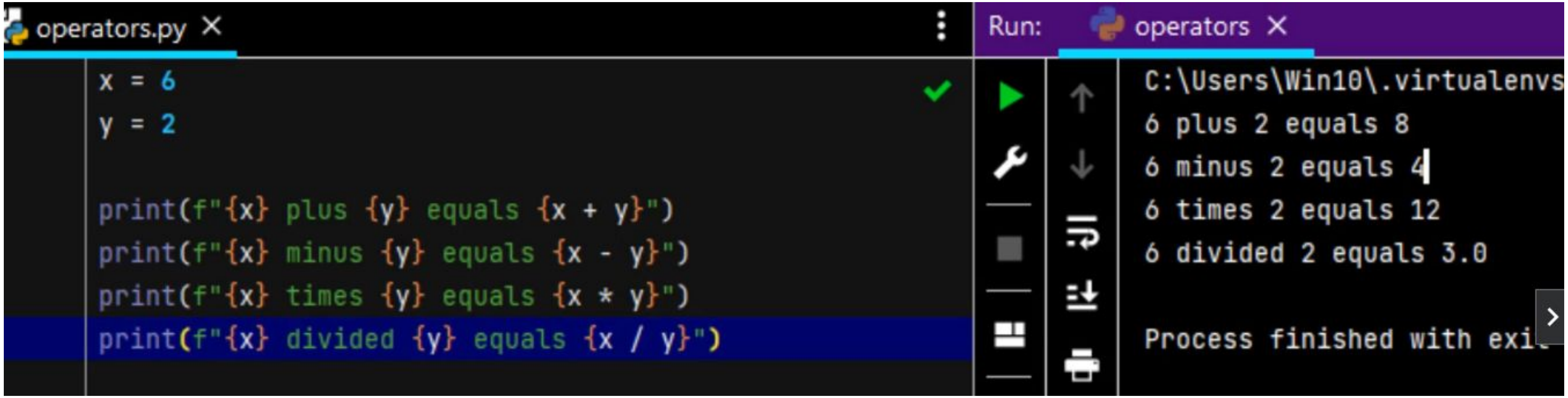
Remember that the precision of your answer (spaces, spelling and punctuation) is very important to arrive at the correct result.

```
previous_points = 875
new_points = 350
```

arithmetic operators

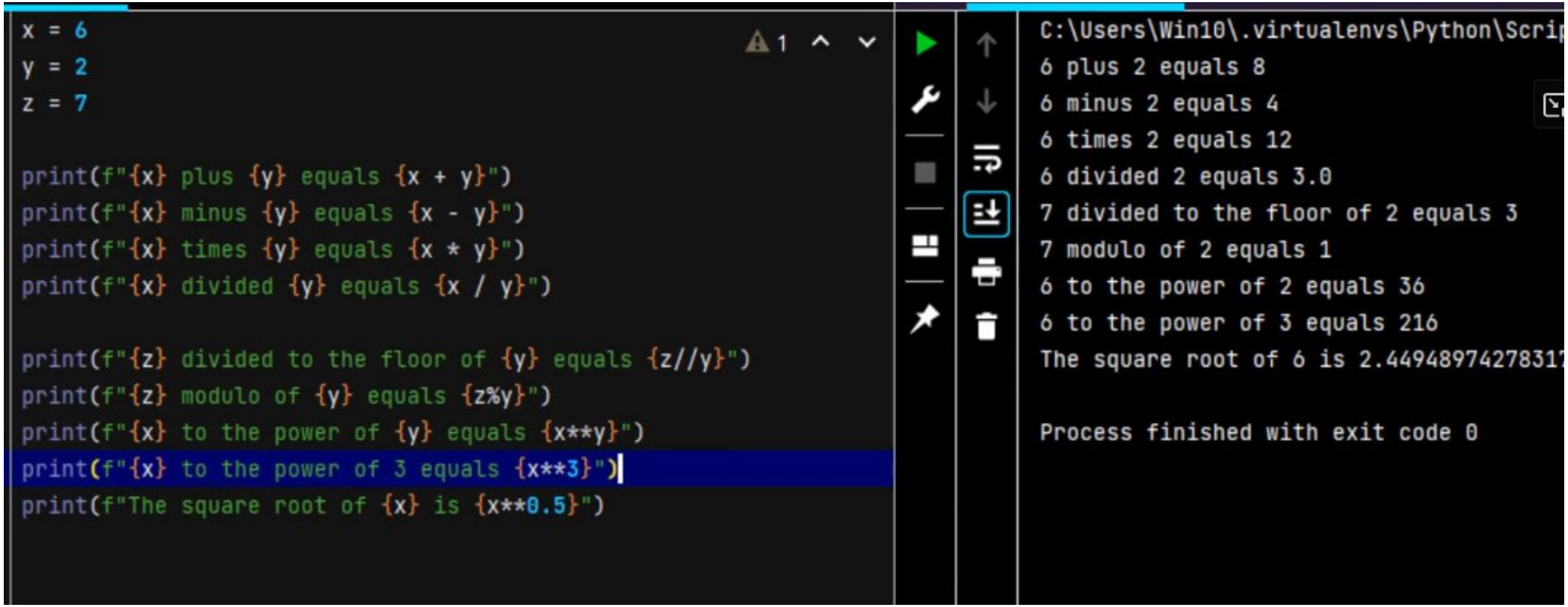
Arithmetic operators are used with numeric values to perform common mathematical operations:

Addition:	`+`	
Subtraction:	`-`	
Multiplication:	`*`	
Division:	`/`	
Floor division:	`//`	
Modulus:	`%`	useful for detecting even values :)
Exponentiation:	`**`	
Square root:	`**0.5`	just a special case of exponentiation



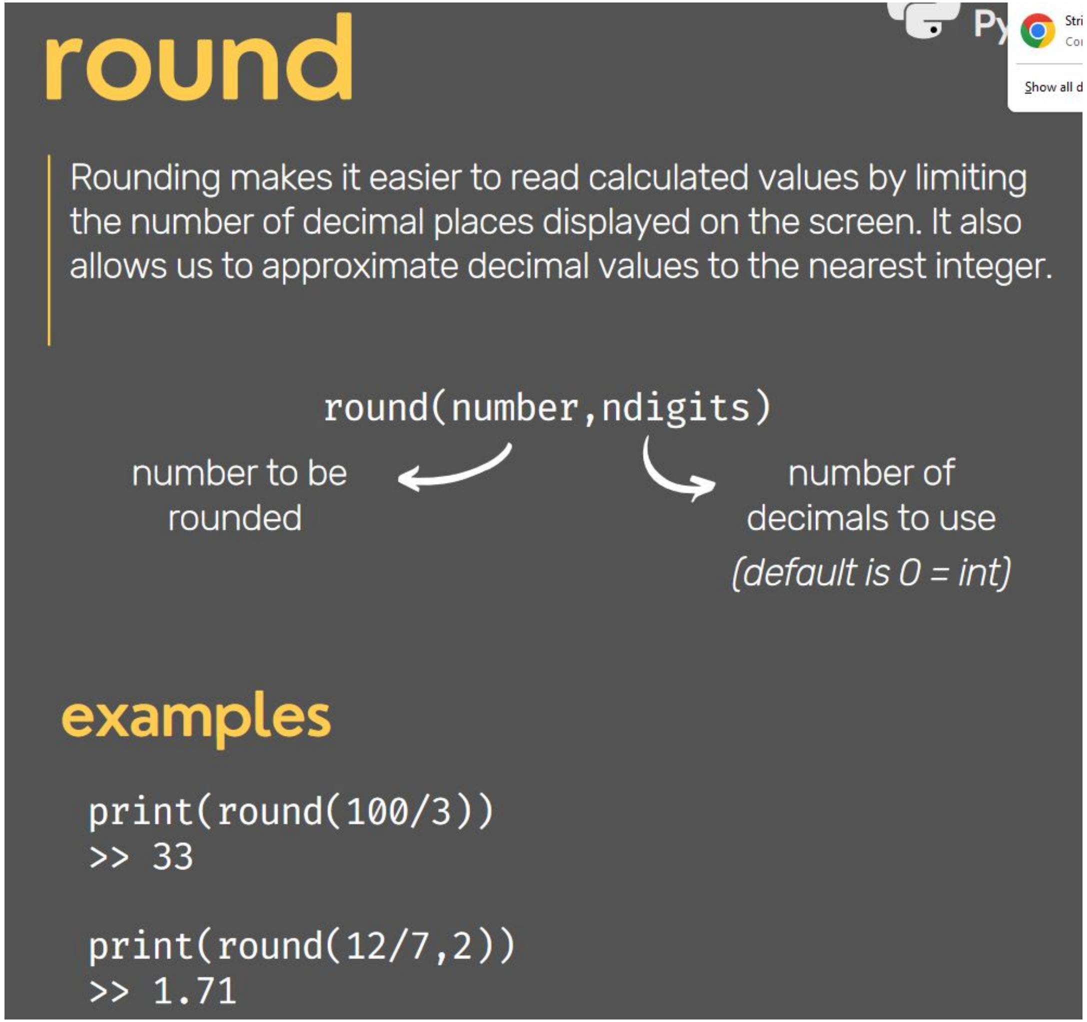
```python
x = 6
y = 2

print(f"{x} plus {y} equals {x + y}")
print(f"{x} minus {y} equals {x - y}")
print(f"{x} times {y} equals {x * y}")
print(f"{x} divided {y} equals {x / y}")
```

```
C:\Users\Win10\.virtualenvs
6 plus 2 equals 8
6 minus 2 equals 4
6 times 2 equals 12
6 divided 2 equals 3.0

Process finished with exi
```



```python
x = 6
y = 2
z = 7

print(f"{x} plus {y} equals {x + y}")
print(f"{x} minus {y} equals {x - y}")
print(f"{x} times {y} equals {x * y}")
print(f"{x} divided {y} equals {x / y}")

print(f"{z} divided to the floor of {y} equals {z//y}")
print(f"{z} modulo of {y} equals {z%y}")
print(f"{x} to the power of {y} equals {x**y}")
print(f"{x} to the power of 3 equals {x**3}")
print(f"The square root of {x} is {x**0.5}")
```

```
C:\Users\Win10\.virtualenvs\Python\Scrip
6 plus 2 equals 8
6 minus 2 equals 4
6 times 2 equals 12
6 divided 2 equals 3.0
7 divided to the floor of 2 equals 3
7 modulo of 2 equals 1
6 to the power of 2 equals 36
6 to the power of 3 equals 216
The square root of 6 is 2.44948974278317

Process finished with exit code 0
```



round

Rounding makes it easier to read calculated values by limiting the number of decimal places displayed on the screen. It also allows us to approximate decimal values to the nearest integer.

```
round(number,ndigits)
```

number to be rounded

number of decimals to use *(default is 0 = int)*

examples

```
print(round(100/3))
>> 33

print(round(12/7,2))
>> 1.71
```

Day 2 Python Challenge

Congratulations: You've finished learning all the content of **Day 2**, and now you're able to create a slightly more complex program.

You already know how to do things like mathematical operations, data type conversions, string formatting and rounding. Let's celebrate with a new challenge.

The situation is this: You work in a company where salespeople receive 13% commissions on their total sales, and your boss wants you to help the salespeople calculate their commissions by creating a program that asks them their name and how much they've sold this month.

Your program will answer them with a sentence that includes their name and the amount of commissions they are entitled to.

Now, this is not a complex program, but that's okay, you're still learning. Even though what you have learned so far is very simple, putting it all together in one program can make your new skills get a little bit messy.

A couple of guidelines:

- This program should start by asking the user for things. So, you're going to need to use input to receive user input, and you should use variables to store that input. Remember that user inputs are stored as strings, so you should convert one of those inputs to float to be able to do operations on it.
- And what operation do you need to do? Well, calculate 13% of the number that the user has entered. That is, multiply the number by 13, then divide by 100.
- It would be good to print the result on the screen, so make sure this information has no more than two decimals, so it is going to be easy to read. Then, organize all that in a string that you would format. Remember, we learned two ways to do it, and any of them is valid.

All right, go on. Try to solve it. And if it gets complicated, Federico will do it with you in the next lesson.

Go ahead and good luck.