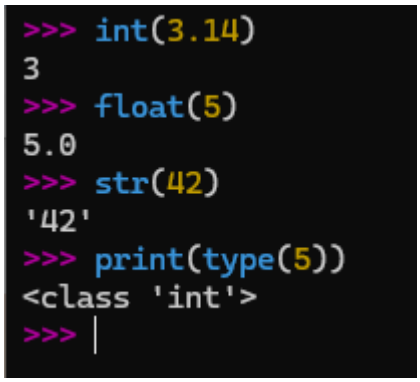
Chapter 2: Basic Syntax

2.1 Variables and data types

Declare without a keyword:
age = 30 (int, no decimals),
weight = 70.5 (float),
name = "Ali" or 'Ali' (str, single/double/triple quotes for multi-line strings),
true_value = True (bool, capitalized).

Use type(age) to check;
convert with int(3.14), float(5), str(42).



Example:
counter = 5;
print(type(counter)) displays <class 'int'>.

2.2 Basic operations

Arithmetic operators
Arithmetic operations in Python work with int (integers) and float (decimal numbers), following the PEMDAS order of operations (parentheses, exponents, multiplication/division, addition/subtraction).

+ : addition
- : subtraction
* : multiplication
/ : true division (always returns a float)
// : integer (floor) division
% : modulo (remainder)

** : exponentiation (power)

Examples with int only

```
python
a = 10  # int
b = 3   # int
print(a + b)   # 13 (int)
print(a - b)   # 7 (int)
print(a * b)   # 30 (int)
print(a / b)   # 3.333... (float!)
print(a // b)  # 3 (int)
print(a % b)   # 1 (int)
print(a ** b)  # 1000 (int)
```

Examples with float

```
python
x = 10.0  # float
y = 3.0   # float

print(x + y)   # 13.0 (float)
print(x / y)   # 3.333... (float)
print(x // y)  # 3.0 (float)
print(x % y)   # 1.0 (float)
```

Operator precedence: PEMDAS (Parentheses, Exponents, Multiplication/Division, Addition/Subtraction)

Example 1: Parentheses first

```
python
print((2 + 3) * 4)  # 20 (addition, then multiplication)
print(2 + 3 * 4)    # 14 (multiplication, then addition)
```

Example 2: Exponents

```
python
print(2 * 3**2)     # 18 (3**2 = 9, then 2*9)
```

Example 3: Multiplication/Division/Modulo (left to right)

```
python
print(10 / 2 * 3)   # 15.0 (10/2 = 5.0, then 5.0*3)
print(10 % 3)       # 1 (remainder of 10/3)
print(10 // 3)      # 3 (integer division)
```

Comparisons: == equal, != not equal, < less, > greater, <=, >=
Logical operators: and (both true), or (at least one true), not (negation)

Example

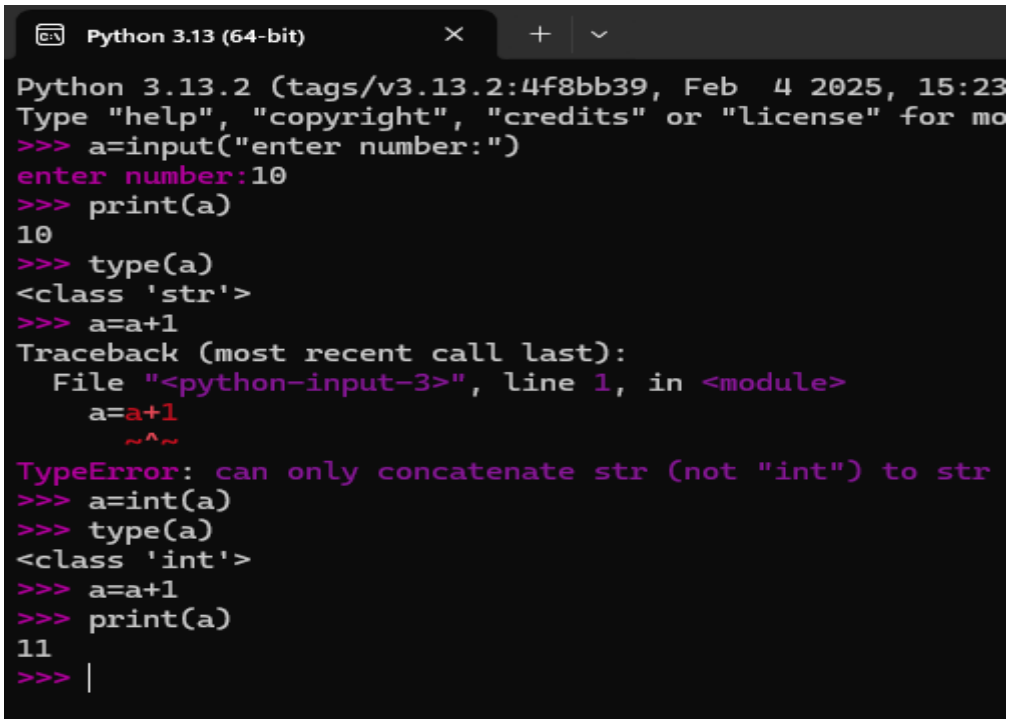
```
python
x = 10
y = 5
print(x > y and x != 0)   # True
```

Input function in Python

The **input**() function reads data from the users of our program (similar to a read() function in algorithms).

input() reads everything the user types as text (a string). So, if we want to do arithmetic with numbers entered by the user, we can't do it directly, because even numbers are read as text.

To solve this, Python provides functions that convert text into an integer or a real number (e.g., int() and float()).



Examples

```
python
# 1
print("enter any text you want")
text = input()
print(text)
# 2
print("enter an integer number")
x = int(input())
print(x)
# 3
```

```
x = int(input("enter an integer number: "))
print(x)
# 4
print("enter a real number")
y = float(input())
print(y)
```

print() function in python

Ex1

```
1>>> x = 32
2 >>> name = "Ahmed"
3 >>> print(name, "is", x, "years old")
4 Ahmed  is  32 years old

5 >>> print(name, "is", x, "years old", sep="")
6 Ahmedis32 years old

7 >>> print(name, "is", x, "years old ", sep="-")
8 Ahmed-is-32- years old

9 >>> print(name, "is", x, "years old ", sep="_")
10 Ahmed_is_32_years old
```

Ex2

```
1 >>> a = "Bon"
2 >>> b = "jour"
3 >>> print(a, b)
4 Bon jour
5 >>> print(a + b)
6 Bonjour
7 >>> print(a, b, sep="")
8 Bonjour
```

Ex3

```
1 >>> x = 32
2 >>> name = "Ahmed"
3 >>> print(f"{name} is {x} years old")
4 Ahmed is  32 years old
```

2.3 Control Structures

2.3.1 Conditional structures

Here is the syntax of the conditional structures if, if…else, and if…elif…else in Python, with simple examples.

1. Simple condition: **if**

Execute a block of code only if a condition is true.

```
python
if condition:
    # block executed if the condition is true
    instruction1
    instruction2
```

Example:

```
python
x = 10
if x > 5:
    print("x is greater than 5")
```

2. Condition with "else": if ... else

Choose between two blocks of code: one if the condition is true, another otherwise.

```
python
if condition:
    # block if the condition is true
    instruction1
else:
    # block if the condition is false
    instruction2
```

Example:

```
python
age = 16
if age >= 18:
    print("Adult")
else:
    print("Minor")
```

3. Multiple cases: if ... elif ... else

Test several conditions one after another.

```
python
if condition1:
    # block if condition1 is true
    instruction1
elif condition2:
    # block if condition2 is true (if condition1 was false)
    instruction2
elif condition3:
    # block if condition3 is true (if the previous ones were
false)
    instruction3
else:
    # block if none of the conditions is true
    instruction4
```

Example:

```
python
grade = 13
if grade >= 16:
    print("Very good")
elif grade >= 14:
    print("Good")
elif grade >= 10:
    print("Satisfactory")
else:
    print("Insufficient")
```

Python stops at the first true test, then exits the structure.

Important syntax points

- Always put `:` after `if`, `elif`, and `else`.
- Always indent the block (usually 4 spaces) under `if`, `elif`, and `else`.
- There is never a condition after `else`.
- Conditions use comparison operators: ==, `!=`, <, >, <=, >= and logical operators `and`, `or`, `not`.

Combined example:

```
python
x = 7
if x > 10 and x < 20:
    print("Between 10 and 20")
elif x == 10:
    print("Equal to 10")
else:
    print("Other case")
```

2.3.2 Repetitive structures

For loop syntax

```
Iterates over a sequence (list, range, string) until the end.
python
for variable in sequence:
    # indented block (4 spaces)
    instructions
```

Examples:

```
python
# Counter from 0 to 4
for i in range(5):
    print(i)
```

```
# Result: 0 1 2 3 4

# Over a list
fruits = ["apple", "banana", "orange"]
for fruit in fruits:
    print(fruit)
# Result:
# apple
# banana
# orange

# range with start, stop, step
for i in range(2, 10, 2):  # 2, 4, 6, 8
    print(i)


n = 3
print("I am going to ask you for", n, "numbers")
for i in range(n):
    x = int(input("Enter a number: "))
    if x > 0:
        print(x, "is positive")
    else:
        print(x, "is negative or zero")
print("End")
```

While loop syntax (Python)

Repeats as long as the condition remains true.

```
python
while condition:
    # indented block
    instructions
```

Example:

```
python
# Simple counter
i = 0
while i < 5:
    print(i)
    i += 1  # Required to avoid an infinite loop!
# Output: 0 1 2 3 4
```

Sum up to a threshold

```
total = 0
while total < 10:
```

```
    total += 2
    print(total)
# Output: 2 4 6 8 10
```

Controls in loops

- **`break`**: exit immediately.
- **`continue`**: skip to the next iteration.
- **`else`**: runs if there was no `break`.

```
python
for i in range(10):
    if i == 5:
        break      # Stops at 5
    print(i)
# Output: 0 1 2 3 4
```