# Introduction to


Martin Malý
Struktura 2024

# Ahoj!

What’s your name?

What’s your field of science?

Have you coded before?

What do you like?

# Acknowledgement


- University of Southampton
- CCP4
  Integrated suite of programs for macromolecular crystallography

- Slides mostly from the Open Educational Resources, University of Edinburgh
  *https://open.ed.ac.uk/introduction_to_python/*

# Content

- Why Python
- First code
- Variables and their types
- Arithmetic and comparison operators
- if elif else
- Lists
- for and while loops
- Definition of functions
- NumPy and Flex arrays
- sys module

# Stop me and ask!

# Many little exercises during our session

# Why to use Python in science?

- Now one of the most important languages for:
  - Data science
  - Machine learning
  - General software development
- Programming automation and pipelines – can save your time
- More powerful alternative for MS Excel
- Libraries: **CCTBX, GEMMI, Coot headless API (CHAPI)**,
  NumPy, Pandas, MatPlotLib, SciPy, scikit-learn, statsmodels…


https://www.sportsrec.com/7662906/the-dangers-of-jumping-into-water-from-heights

# Crystallographic restriction theorem

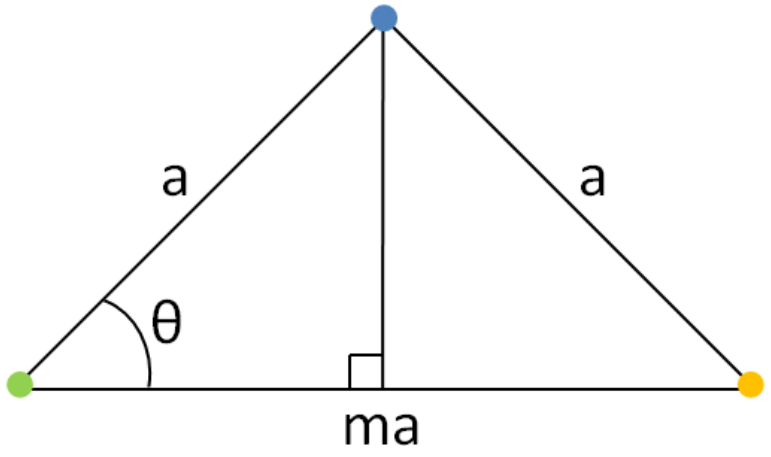
The rotational symmetries of a crystal are (usually) limited to 2-fold, 3-fold, 4-fold, and 6-fold.

$$2a\cos\theta = 2a\cos\frac{2\pi}{n}.$$

Equating the two relations gives:

$$2\cos\frac{2\pi}{n} = m$$

This is satisfied by only $n = 1, 2, 3, 4, 6$.




- *https://en.wikipedia.org/wiki/Crystallographic_restriction_theorem*

```python
import math

def check_symmetry(maximum):
    result = []
    for i in range(maximum):
        if i == 0:
            print("It does not have sense to test the 0-fold symmetry.")
            continue  # continue with the next i
        else:
            value = 2 * math.cos(2 * math.pi / i)
        if math.isclose(value, round(value), abs_tol=0.001): # if value is an integer
            print(str(i) + "-fold symmetry is possible.")
            result.append(i)
        else:
            print(str(i) + "-fold symmetry is not possible.")
    print("Result: possible symmetry:")
    print(result)
    return result

check_symmetry(10)
```

```
It does not have sense to test the 0-fold symmetry.
1-fold symmetry is possible.
2-fold symmetry is possible.
3-fold symmetry is possible.
4-fold symmetry is possible.
5-fold symmetry is not possible.
6-fold symmetry is possible.
7-fold symmetry is not possible.
8-fold symmetry is not possible.
9-fold symmetry is not possible.
Result: possible symmetry:
[1, 2, 3, 4, 6]
[1, 2, 3, 4, 6]
```

$$2\cos\frac{2\pi}{n} = m$$

```python
import math

def check_symmetry(maximum):
    result = []
    for i in range(maximum):
        if i == 0:
            print("It does not have sense to test the 0-fold symmetry.")
            continue  # continue with the next i
        else:
            value = 2 * math.cos(2 * math.pi / i)
        if math.isclose(value, round(value), abs_tol=0.001): # if value is an integer
            print(str(i) + "-fold symmetry is possible.")
            result.append(i)
        else:
            print(str(i) + "-fold symmetry is not possible.")
    print("Result: possible symmetry:")
    print(result)
    return result

check_symmetry(10)
```

```
It does not have sense to test the 0-fold symmetry.
1-fold symmetry is possible.
2-fold symmetry is possible.
3-fold symmetry is possible.
4-fold symmetry is possible.
5-fold symmetry is not possible.
6-fold symmetry is possible.
7-fold symmetry is not possible.
8-fold symmetry is not possible.
9-fold symmetry is not possible.
Result: possible symmetry:
[1, 2, 3, 4, 6]
[1, 2, 3, 4, 6]
```

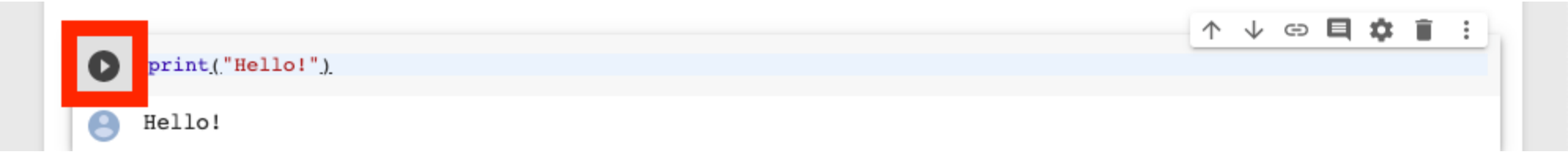
$$2\cos\frac{2\pi}{n} = m$$

**What can we see?**

- Import of a library
- Definition of a function
- Indentation
- Square brackets… a list
- Variable
- for loop
- 0 is the 1st number
- if and else clause
- Comparing ==
- print()
- Comment
- Function from a library
- Function parameters
- Data types
- Calling a function

# How to install and run Python?

- **Google Colab** – no need to install, our solution for today!
  *https://colab.research.google.com*
  Run a cell with a code using a “play” button or Ctrl+Enter




- Another option: Download and install Python
  *https://www.python.org/downloads*
  (not necessary for our session today)

1. **IPython** – Python can be run interactively
2. **Python scripts** – Then we must write text files in .py

# Variables

- Let’s define a variable
- Later it can be called by the variable name
- Variable names are case sensitive and unique

```
distanceToLondonMiles = 403
mileToKm = 1.60934
distanceToLondonKm = distanceToLondonMiles * mileToKm
distanceToLondonKm
```

```
648.56402
```

# Variables

- Let’s define a variables
- Later it can be called by the variable name
- Variable names are case sensitive and unique

```
distanceToLondonMiles = 403
mileToKm = 1.60934
distanceToLondonKm = distanceToLondonMiles * mileToKm
distanceToLondonKm
```

```
648.56402
```

We can now reuse the variable mileToKm in the next block without having to define it again!

```
marathonDistanceMiles = 26.219
marathonDistanceKm = marathonDistanceMiles * mileToKm
print(marathonDistanceKm)
```

```
42.19528546
```

# Printing

- When writing scripts, your outcomes aren't printed on the terminal.
- Thus, you must print them yourself with the print() function.
- Beware to not mix up the different type of variables!

```
print("Python is powerful!")
```

```
Python is powerful!
```

```
x = "Python is powerful"
y = " and versatile!"
print(x + y)
```

```
Python is powerful and versatile!
```

# Task for you


iDNES.cz

ZPRÁVY > KRAJE > HRADEC KRÁLOVÉ

Česko postihne silný vítr. Sněžku zasáhl orkán o rychlosti 124 kilometrů za hodinu

9. března 2024 9:53, aktualizováno 12:51

Convert a wind speed 124 km/h to m/s.

3.6 km/h = 1 m/s

# Task for you


iDNES.cz

ZPRÁVY > KRAJE > HRADEC KRÁLOVÉ

Česko postihne silný vítr. Sněžku zasáhl orkán o rychlosti 124 kilometrů za hodinu

9. března 2024 9:53, aktualizováno 12:51

Convert a wind speed 124 km/h to m/s.

3.6 km/h = 1 m/s

- Define a variable `speed_kmperhour`
- Calculate the converted speed in a variable `speed_mpers`
- Show the result using `print(speed_mpers)`

# Task for you


iDNES.cz

ZPRÁVY > KRAJE > HRADEC KRÁLOVÉ

Česko postihne silný vítr. Sněžku zasáhl orkán o rychlosti 124 kilometrů za hodinu

9. března 2024 9:53, aktualizováno 12:51

Convert a wind speed 124 km/h to m/s.

3.6 km/h = 1 m/s

- Define a variable `speed_kmperhour`
- Calculate the converted speed in a variable `speed_mpers`
- Show the result using `print(speed_mpers)`

```
speed_kmperhour = 124
speed_mpers =
```

# Task for you


iDNES.cz

ZPRÁVY > KRAJE > HRADEC KRÁLOVÉ

Česko postihne silný vítr. Sněžku zasáhl orkán o rychlosti 124 kilometrů za hodinu

9. března 2024 9:53, aktualizováno 12:51

Convert a wind speed 124 km/h to m/s.

3.6 km/h = 1 m/s

- Define a variable `speed_kmperhour`
- Calculate the converted speed in a variable `speed_mpers`
- Show the result using `print(speed_mpers)`

0s

```
speed_kmperhour = 124
speed_mpers = 124 / 3.6
print(speed_mpers)
```

```
34.44444444444444
```

# Types

Variables actually have a type, which defines the way it is stored.

The basic types are:

| Type | Declaration | Example | Usage |
|---|---|---|---|
| Integer | int | `x = 124` | Numbers without decimal point |
| Float | float | `x = 124.56` | Numbers with decimcal point |
| String | str | `x = "Hello world"` | Used for text |
| Boolean | bool | `x = True` or `x = False` | Used for conditional statements |
| NoneType | None | `x = None` | Whenever you want an empty variable |

```
In [4]: x = 10    # This is an integer
        y = "20"  # This is a string
        x + y
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-4-f1463b8b4c2e> in <module>()
      1 x = 10    # This is an integer
      2 y = "20"  # This is a string
----> 3 x + y

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

**Important lesson to remember!**
We can't do arithmetic operations on variables of different types. Therefore make sure that you are always aware of your variables types!

You can find the type of a variable using **type()**. For example type **type(x)**.

# Casting types

Luckily Python offers us a way of converting variables to different types!

Casting – the operation of converting a variable to a different type

```
x = 10    # This is an integer
y = "20"  # This is a string
x + int(y)
```

```
30
```

Similar methods exist for other data types: **int()**, **float()**, **str()**

```
print(str(i) + "-fold symmetry is possible.")
```

# Another more generic way to fix it

```
str1 = "It has"
str2 = 76
str3 = "methods!"
print(str1, str2, str3)
```

```
It has 76 methods!
```

If we comma separate statements in a print function we can have different variables printing!

# Arithmetic operations

Similar to actual Mathematics.

Order of precedence is the same as in Mathematics.

We can also use parenthesis ()

| Symbol | Task Performed | Example | Result |
|---|---|---|---|
| + | Addition | 4 + 3 | 7 |
| - | Subtraction | 4 - 3 | 1 |
| / | Division | 7 / 2 | 3.5 |
| % | Mod | 7 % 2 | 1 |
| * | Multiplication | 4 * 3 | 12 |
| // | Floor division | 7 // 2 | 3 |
| ** | Power of | 7 ** 2 | 49 |

# Comparison operators

- Return Boolean values (i.e. True or False)
- Used extensively for conditional statements (if)

| Operator | Output |
|---|---|
| x == y | True if x and y have the same value |
| x != y | True if x and y don't have the same value |
| x < y | True if x is less than y |
| x > y | True if x is more than y |
| x <= y | True if x is less than or equal to y |
| x >= y | True if x is more than or equal to y |

# Comparison operators

- Return Boolean values (i.e. True or False)
- Used extensively for conditional statements (if)

```
if i == 0:
    print("It does not have sense to test the 0-fold symmetry.")
    continue  # continue with the next i
else:
    value = 2 * math.cos(2 * math.pi / i)
if math.isclose(value, round(value), abs_tol=0.001): # if value is
    print(str(i) + "-fold symmetry is possible.")
    result.append(i)
else:
    print(str(i) + "-fold symmetry is not possible.")
```

| Operator | Output |
|---|---|
| x == y | True if x and y have the same value |
| x != y | True if x and y don't have the same value |
| x < y | True if x is less than y |
| x > y | True if x is more than y |
| x <= y | True if x is less than or equal to y |
| x >= y | True if x is more than or equal to y |

# Logical operators

- Allows us to extend the conditional logic
- Will become essential later on

| Operation | Result |
|---|---|
| x or y | True if at least on is True |
| x and y | True only if both are True |
| not x | True only if x is False |

| a | not a |
|---|---|
| False | True |
| True | False |

| a | b | a and b | a or b |
|---|---|---|---|
| False | False | False | False |
| False | True | False | True |
| True | False | False | True |
| True | True | True | True |

*Truth-table definitions of bool operations*

# Strings

```
x = "Python"
y = "rocks"
x + " " + y
```

```
'Python rocks'
```

# Multiline strings

```
x = """To include
multiple lines
you have to do this"""
y ="or you can also\ninclude the special\ncharacter `\\n` between lines"
print(x)
print(y)
```

```
To include
multiple lines
you have to do this
or you can also
include the special
character `\n` between lines
```

# if elif else

- Fundamental building block of software

```
print("Let's divide two values")
nom = 2
denom = 0  # try different values
if denom == 0:
    print("Division by zero is not possible in this universe!")
elif denom == 1:
    print("Division by one is possible but quite useless...")
    print("The result is " + str(nom))
else:
    result = nom / denom
    print("The result is " + str(result))
```

**Conditional statement**

**Executed if answer is True**

# Indentation matters!

- Code is grouped by its indentation
- Indentation is the number of whitespace or tab characters before the code.
- If you put code in the wrong block then you will get unexpected behaviour

```
x = 10
if x%2 == 0:
    print(x,'is even!')
    if x%5 == 0:
        print(x,'is divisible by 5!')
        print('Output only when x is divisible by both 2 and 5.')
    else:
        print(x,'is not divisible by 5!')
        print('Output only when x is divisible by 2 but not divisible by 5.')
else:
    print(x,'is odd!')
print('No indentation. Output in all cases.')
```

```
10 is even!
10 is divisible by 5!
Output only when x is divisible by both 2 and 5.
No indentation. Output in all cases.
```

# Task for you

Let's define a variable `age`.

Write an if-elif-else clause depending on the `age`:

- If the `age` is below 18, print "Beer not allowed".
- If the `age` is above 18 (inclusively) and below 21, print "Beer allowed in Czechia but not in America".
- If the `age` is above 21 (inclusively), print "Beer allowed".

# Lists

- One of the most useful concepts
- Group multiple variables together (a kind of **container**!)

```
fruits = ["apple", "orange", "tomato", "banana"]  # a list of strings
print(type(fruits))
print(fruits)
```

```
<class 'list'>
['apple', 'orange', 'tomato', 'banana']
```

# Indexing a list

- Indexing – accessing items within a data structure

```
fruits[2]
```

```
'tomato'
```

- Indexing a list is not very intuitive…
- The first element of a list has an **index 0**

| Index: | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| List: | apple | orange | tomato | banana |

# Quick quiz

What will **fruits[3]** return?

```
fruits = ["apple", "orange", "tomato", "banana"]  # a list of strings
print(type(fruits))
print(fruits)
```

```
<class 'list'>
['apple', 'orange', 'tomato', 'banana']
```

# Is a tomato really a fruit?

```
fruits[2] = "apricot"
print(fruits)
```

```
['apple', 'orange', 'apricot', 'banana']
```

Furthermore, we can modify lists in various ways

```
fruits.append("lime")    # add new item to list
print(fruits)
fruits.remove("orange")  # remove orange from list
print(fruits)
```

```
['apple', 'orange', 'apricot', 'banana', 'lime']
['apple', 'apricot', 'banana', 'lime']
```

# Lists with integers

***range()*** - a function that generates a sequence of numbers as a list

```
nums = list(range(10))
print(nums)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
nums = list(range(0, 100, 5))
print(nums)
```

```
[0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85,
90, 95]
```

# Slicing lists

- Slicing – obtain a particular set of sub-elements from a data structure.
- Very useful and flexible.

```
print(nums)
print(nums[1:5:2]) # Get from item 1(starting point) through item 5(end point, not included) with step size 2
print(nums[0:3]) # Get items 0 through 3(not included)
print(nums[4:]) # Get items 4 onwards
print(nums[-1]) # Get the last item
print(nums[::-1]) # Get the whole list backwards
```

```
[0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95]
[5, 15]
[0, 5, 10]
[20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95]
95
[95, 90, 85, 80, 75, 70, 65, 60, 55, 50, 45, 40, 35, 30, 25, 20, 15, 10, 5, 0]
```

# Lists – helpful functions

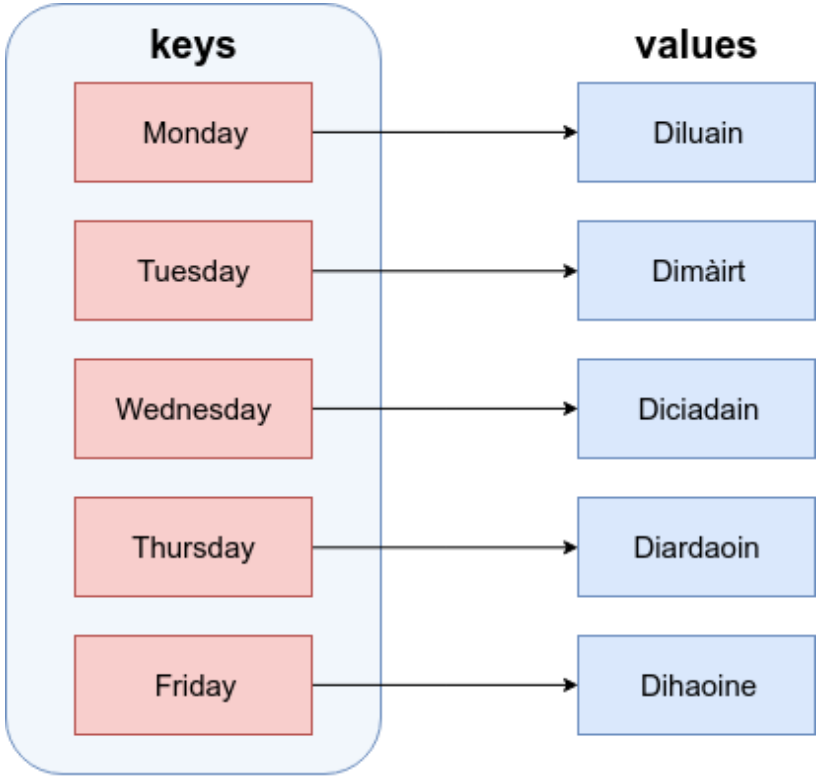
- Makes them extremely useful and versatile

```
nums = list(range(0, 100, 5))
print(nums)
print(len(nums))   # number of items within the list
print(max(nums))   # the maximum value within the list
print(min(nums))   # the minimum value within the list
print(sum(nums))   # sum of the values
print(sum(nums) / len(nums)) # the average value
```

```
[0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95]
20
95
0
950
47.5
```

# Dictionaries

- Similar to actual dictionaries
- They are effectively 2 lists combined – keys and values
- We use the keys to access the values instead of indexing them like a list
- Each value is mapped to a unique key




```
days = {"Monday": "Diluain", "Tuesday": "Dimàirt",
        "Wednesday": "Diciadain", "Thursday": "Diardaoin",
        "Friday": "Dihaoine"}
print(type(days))
print(days)
```

```
<class 'dict'>
{'Monday': 'Diluain', 'Tuesday': 'Dimàirt', 'Wednesday': 'Diciadain',
'Thursday': 'Diardaoin', 'Friday': 'Dihaoine'}
```

# Accessing a dictionary

Values are accessed by their keys (just like a dictionary)
Note that they can't be indexed like a list

```
days["Friday"]
```

```
'Dihaoine'
```

It is possible to obtain only the keys or values of a dictionary.
This is useful for iteration.

```
print(days.keys())   # get only the keys of the dictionary
print(days.values()) # get only the values of the dictionary
```

```
dict_keys(['Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday'])
dict_values(['Dimàirt', 'Diciadain', 'Diardaoin', 'Dihaoine', 'Disathai
rne'])
```

# For loop

- Say we want to go over a list and print each item along with its index

```
fruits = ["apple", "orange", "tomato", "banana"]
print("The fruit", fruits[0], "has index", fruits.index(fruits[0]))
print("The fruit", fruits[1], "has index", fruits.index(fruits[1]))
print("The fruit", fruits[2], "has index", fruits.index(fruits[2]))
print("The fruit", fruits[3], "has index", fruits.index(fruits[3]))
```

```
The fruit apple has index 0
The fruit orange has index 1
The fruit tomato has index 2
The fruit banana has index 3
```

- What if we have much more than 4 items in the list, say, 1000?

# For loop – example

- Now with a for loop

```
fruitList = ["apple", "orange", "tomato", "banana"]
for fruit in fruitList:
    print("The fruit", fruit, "has index", fruitList.index(fruit))
```

```
The fruit apple has index 0
The fruit orange has index 1
The fruit tomato has index 2
The fruit banana has index 3
```

- Saves us writing more lines
- Doesn't limit us in term of size
- Indentation

# Task for you

Define a list of strings – your favourite food.

For each food, print “I like `food`”.

# While loop

- Another useful loop. Similar to the for loop.
- A while loop doesn't run for a predefined number of iterations, like a for loop. Instead, it stops as soon as a given condition becomes true/false.

```
n = 0
while n < 5:
    print("Executing while loop")
    n = n + 1

print("Finished while loop")
```

```
Executing while loop
Executing while loop
Executing while loop
Executing while loop
Executing while loop
Finished while loop
```

# Task for you

Write a code to calculate factorial of 5.

Use a `for` loop or a `while` loop.

5! = 5 * 4 * 3 * 2 * 1 = 120

# Task for you

Write a code to calculate factorial of 5.

Use a `for` loop or a `while` loop.

5! = 5 * 4 * 3 * 2 * 1 = 120

```
fac = 1
for i in range(5):
    fac = fac * (i + 1)
print(fac)
```

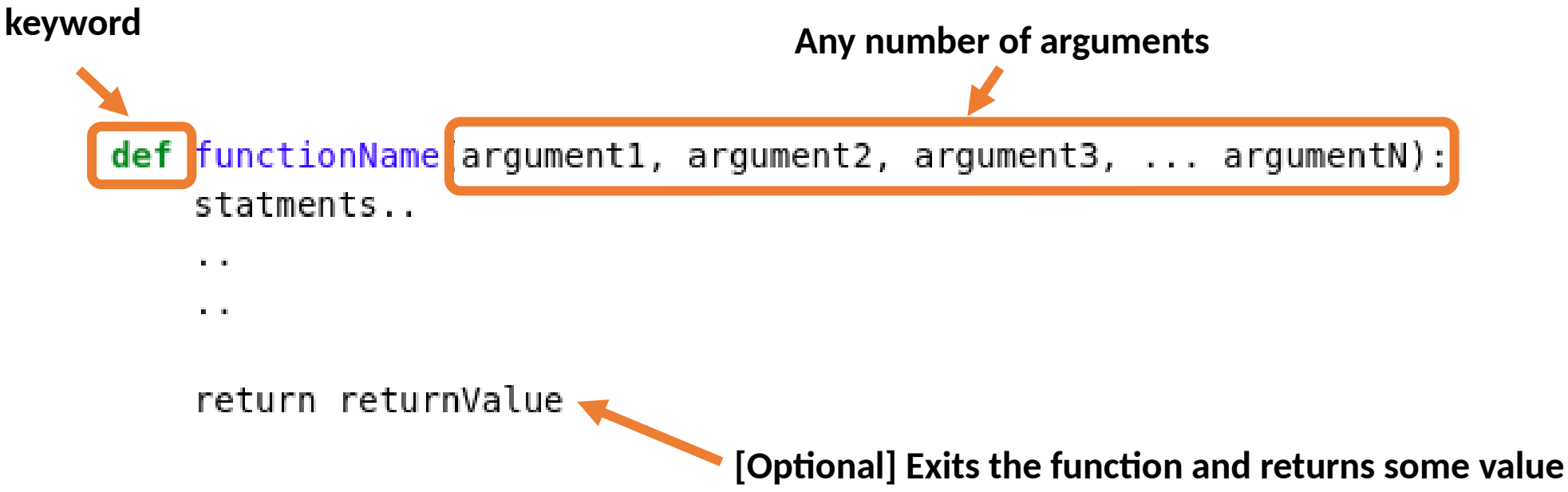
```
120
```

```
fac = 1
i = 1
while i < 5:
    fac = fac * (i + 1)
    i = i + 1
print(fac)
```

```
120
```

# Function declaration

- For a code which is used multiple times
- Make code modular and readable
- Functions accept arguments and execute a piece of code
- Often they also return values (the result of their code)



# Function examples

```
def printNum(num):
    print("My favourite number is", num)

printNum(7)
printNum(14)
printNum(2)
```

```
My favourite number is 7
My favourite number is 14
My favourite number is 2
```

```
def convert_speed(speed_kmperhour):
    speed_mpers = speed_kmperhour / 3.6
    print(speed_mpers)
    return speed_mpers

speed1_converted = convert_speed(124)
speed2_converted = convert_speed(50)
speed3_converted = convert_speed(90)
speed4_converted = convert_speed(130)
```

```
34.44444444444444
13.88888888888889
25.0
36.11111111111111
```

# Task for you

Write a **function** to calculate a factorial and call it to calculate 5!.

# Task for you

Write a **function** to calculate a factorial and call it to calculate 5!.

```
def factorial(n):
    ... # your code
```

```
factorial(5)
```

```
120
```

# Task for you

Write a **function** to calculate a factorial and call it to calculate 5!.

```
def factorial(n):
    fac = 1
    for i in range(n):
        fac = fac * (i + 1)
    print(fac)
    return fac

factorial(5)
```

```
120
```

# Task for you – merging intensities

Imagine that you measured an intensity of the reflection (523)

From 3 different diffraction images, you measured 3 different values:

$I_{523,1} = 2055, \quad I_{523,2} = 1866, \quad I_{523,3} = 2214$

**Write a list of the measured intensities,**
**calculate an average value $I_{523}$ and multiply it by 10.**

# Task for you – merging intensities

Imagine that you measured an intensity of the reflection (523)

From 3 different diffraction images, you measured 3 different values:

$I_{523,1} = 2055, \quad I_{523,2} = 1866, \quad I_{523,3} = 2214$

**Write a list of the measured intensities,**
**calculate an average value $I_{523}$ and multiply it by 10.**

**Calculate a square root of the average intensity.**

You will need to add this line in the beginning: `import math`

So you can use the function `math.sqrt()` in your code.

You will get a merged and scaled amplitude of structure factor $|F_{523}|$ :-)

# NumPy array, Flex array – cool list

- Array with a defined data type.
- Operations can be applied directly for all the elements of an array.
- Can have multidimensional structure.
- Faster calculations.

```
>>> from scitbx.array_family import flex
>>> int_array = flex.int([3,1,2,6,8,2,6,3,4])
>>> double_array = flex.double([(1.5,2,3), (4,5,6)])
>>> print(double_array)
<scitbx_array_family_flex_ext.double object at 0x7f05f76bec20>
>>> list(double_array)
[1.5, 2.0, 3.0, 4.0, 5.0, 6.0]
>>> double_array2 = double_array + 5
>>> list(double_array2)
[6.5, 7.0, 8.0, 9.0, 10.0, 11.0]
>>> double_array2.nd()
2
>>> double_array2.all()
(2, 3)
>>>
```

# sys – system module

- Standard input: `sys.stdin`
- Standard output: `sys.stdout`
- Standard error output: `sys.stderr`

```
import sys
sys.stdout.write("Let's divide two values\n")
nom = 2
denom = 0
if denom == 0:
    sys.stderr.write("ERROR: Division by zero\n")
    sys.exit(1)
result = nom / denom
sys.stdout.write("The result is " + str(result) + "\n")
```

```
Let's divide two values
ERROR: Division by zero
An exception has occurred, use %tb to see the full traceback.

SystemExit: 1
```

# Passing input argument(s) to script

Python script file: greet.py

```
import sys

def greet(name):
    print("Hello " + str(name))

if (__name__ == "__main__"):
    greet(sys.argv[1])
```

Let’s execute the script and specify an argument

```
martin@precision:/tmp$ python3 greet.py Petr
The file name of this script is greet.py
Hello Petr
```