TNG TECHNOLOGY CONSULTING

# Introduction to Python Programming

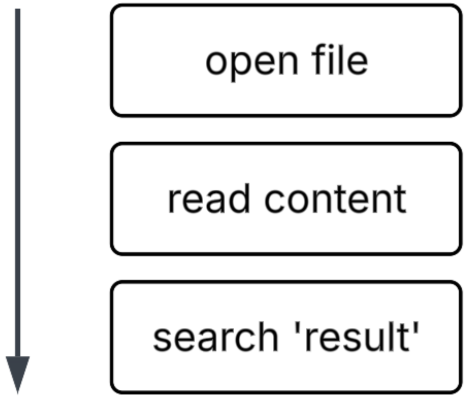
## Workshop @ TUM Graduate School

# Introduction round

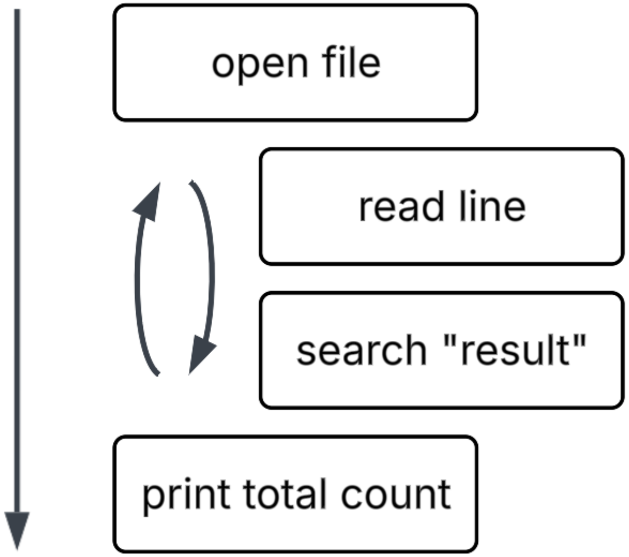
Valentin Zieglmeier - Sebastian Zett

# Basics on programming languages

# Idea: give the computer commands



# Avoiding repetition: loops



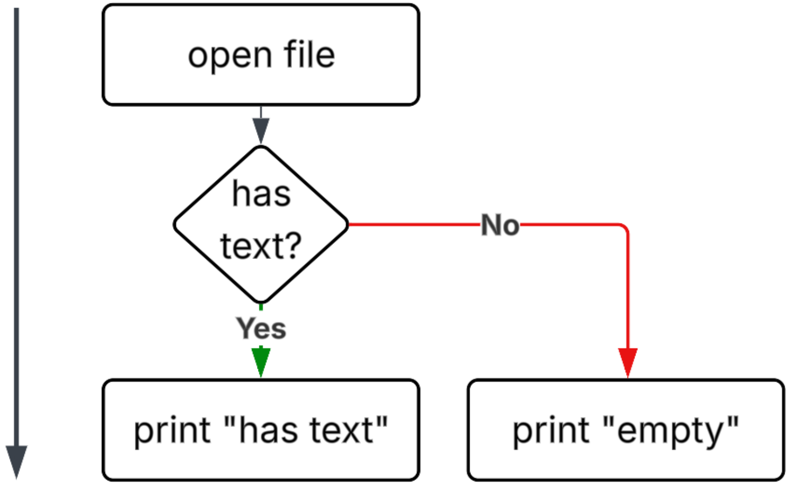
open file

read line

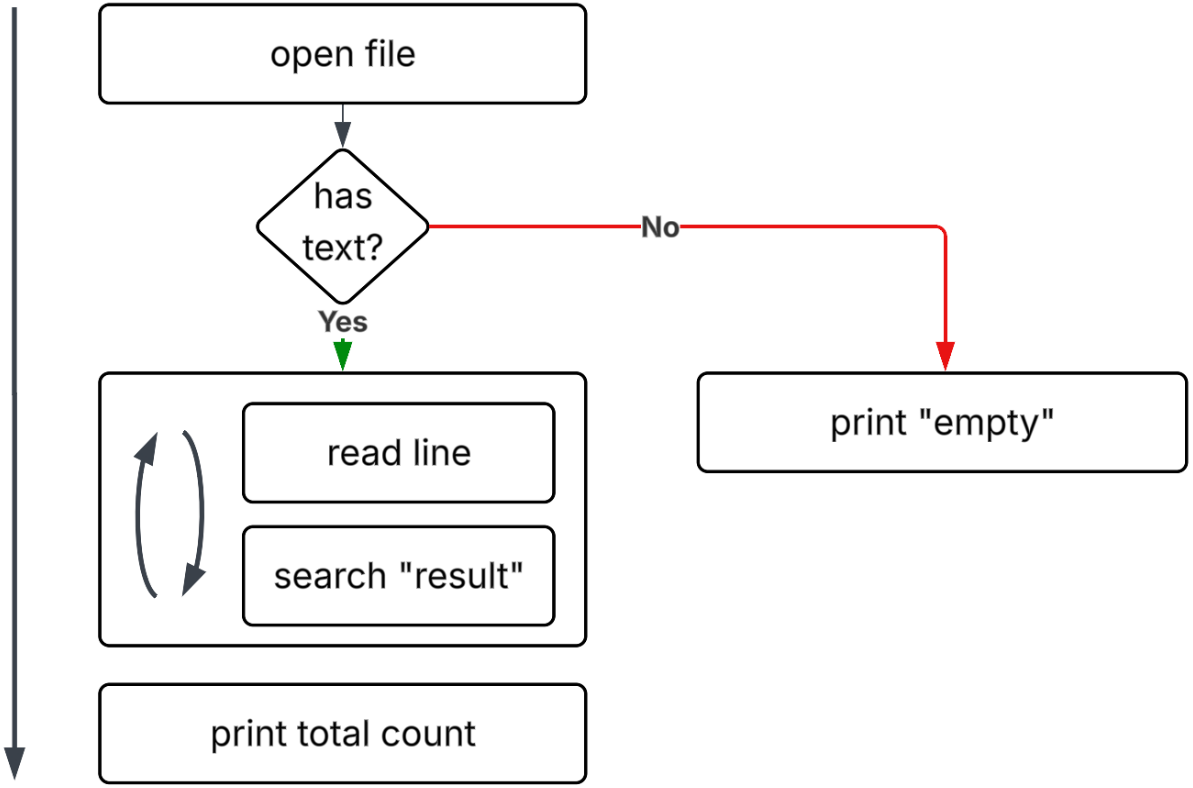
search "result"

print total count

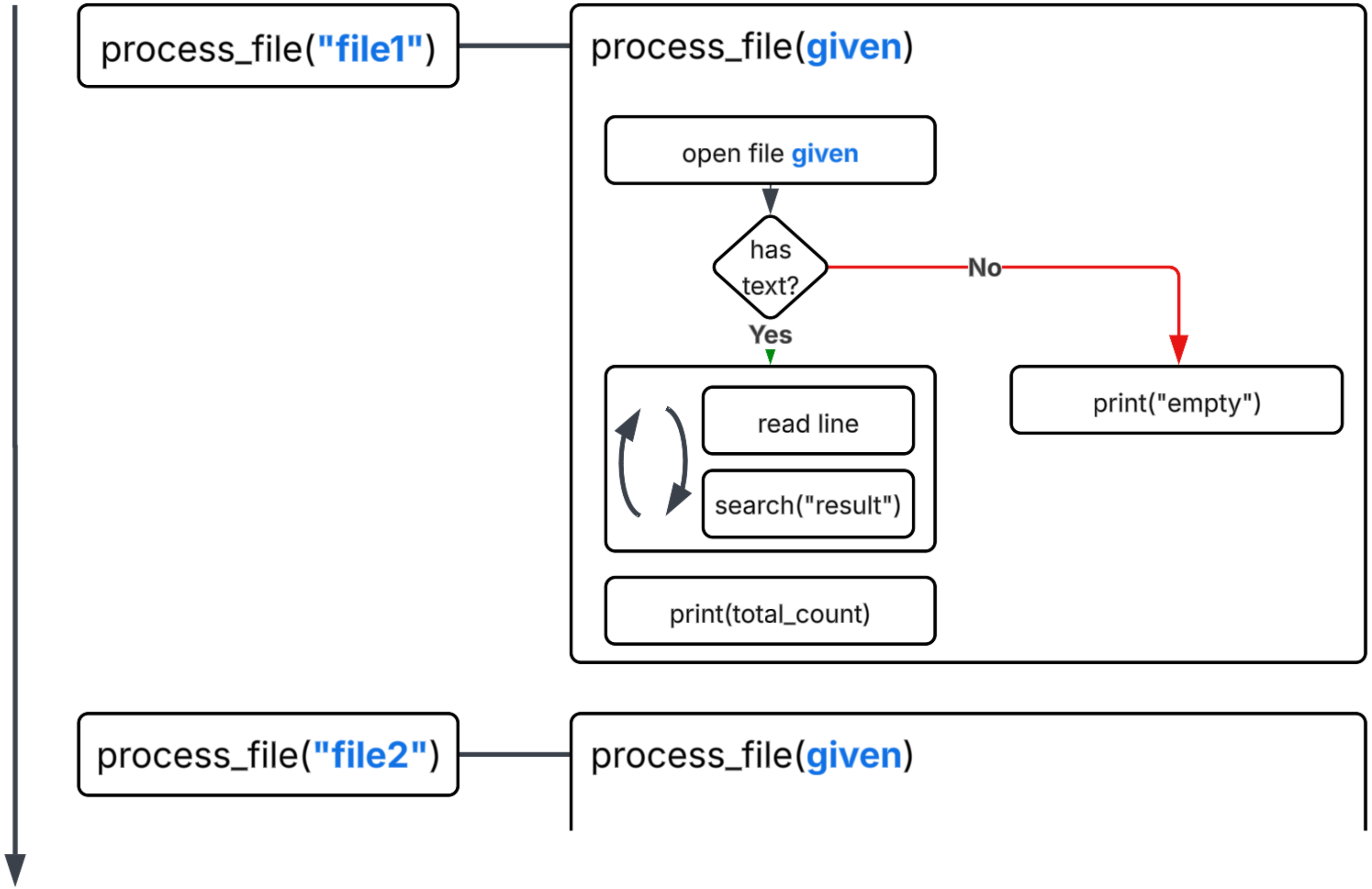
# Test conditions: if



# Combining loops & conditions



# Create reusable code blocks: functions



# Python: the basics

In [1]:

```
import this
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to d
o it.
Although that way may not be obvious at first unless you're Dutc
h.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

# Documentation

https://docs.python.org/3/index.html

# Data types

# Data types

In [2]:

```
# int, float
5
3.5

# bool
True, False

# str
"abc"
print("abc\ndef")
```

```
abc
def
```

# Data types

In [2]:

```
# int, float
5
3.5

# bool
True, False

# str
"abc"
print("abc\ndef")
```

```
abc
def
```

In [3]:

```
n = 1
f = "f"
print(f"{n} interpolated {f}-string")
```

```
1 interpolated f-string
```

# Variables

# Variables

In [4]:

```
# Assignment
x = 5  # integer
name = "Alice"  # string
is_valid = True  # boolean
```

# Variables

In [4]:

```
# Assignment
x = 5  # integer
name = "Alice"  # string
is_valid = True  # boolean
```

In [5]:

```
# Usage
print(name)
```

```
Alice
```

# Variables

In [4]:

```
# Assignment
x = 5  # integer
name = "Alice"  # string
is_valid = True  # boolean
```

In [5]:

```
# Usage
print(name)
```

```
Alice
```

In [6]:

```
# Dynamic typing
print(type(x))
x = "hello"
print(type(x))
```

```
<class 'int'>
<class 'str'>
```

# Operators

# Operators

In [7]:

```
# Arithmetic operators
print(5 + 2)
print(5 - 2)
print(5 * 2)
print(5 / 2)
print(5**2)
```

```
7
3
10
2.5
25
```

# Operators

In [7]:

```
# Arithmetic operators
print(5 + 2)
print(5 - 2)
print(5 * 2)
print(5 / 2)
print(5**2)
```

```
7
3
10
2.5
25
```

In [8]:

```
# Comparison operators
print(5 > 3)
print(5 <= 3)
print(5 == 3)
print(5 != 3)
```

```
True
False
False
True
```

# Operators

In [7]:

```
# Arithmetic operators
print(5 + 2)
print(5 - 2)
print(5 * 2)
print(5 / 2)
print(5**2)
```

```
7
3
10
2.5
25
```

In [8]:

```
# Comparison operators
print(5 > 3)
print(5 <= 3)
print(5 == 3)
print(5 != 3)
```

```
True
False
False
True
```

In [9]:

```
# Logical operators
```

```
print(True or False)
print(True and False)
print(not True)
```

```
True
False
False
```

# First calculations (1/2)

# First calculations (1/2)

```
In [10]: hours = 2
         minutes = 12
         duration_in_minutes = hours * 60 + minutes

         print(duration_in_minutes)
```

```
132
```

# First calculations (1/2)

In [10]:

```
hours = 2
minutes = 12
duration_in_minutes = hours * 60 + minutes

print(duration_in_minutes)
```

```
132
```

In [11]:

```
age = 15
age_of_majority = 18
is_adult = age >= age_of_majority

print(is_adult)
```

```
False
```

# First calculations (2/2)

# First calculations (2/2)

In [12]:

```
is_minor = not is_adult

print(is_minor)
```

```
True
```

# First calculations (2/2)

In [12]:

```
is_minor = not is_adult

print(is_minor)
```

```
True
```

In [13]:

```
# Updating variables
age = 15
age = age + 1
print(age)

age -= 1
print(age)
```

```
16
15
```

# → Lesson #1

# Containers

# Containers

In [14]:

```
my_list = [1, 2, 3]           # empty: [] or list()
my_dict = {"a": 1, "b": 2}  # empty: {} or dict()
```

# Containers

In [14]:

```
my_list = [1, 2, 3]           # empty: [] or list()
my_dict = {"a": 1, "b": 2}  # empty: {} or dict()
```

In [15]:

```
a, b, c = my_list  # sequence unpacking (must have correct count)
print(b)
```

```
2
```

# Containers

```
In [14]: my_list = [1, 2, 3]           # empty: [] or list()
         my_dict = {"a": 1, "b": 2}  # empty: {} or dict()
```

```
In [15]: a, b, c = my_list  # sequence unpacking (must have correct count)
         print(b)

         2
```

```
In [16]: my_dict.keys()

Out[16]:   dict_keys(['a', 'b'])
```

# Containers

```
In [14]: my_list = [1, 2, 3]          # empty: [] or list()
         my_dict = {"a": 1, "b": 2}  # empty: {} or dict()
```

```
In [15]: a, b, c = my_list  # sequence unpacking (must have correct count)
         print(b)
```

```
2
```

```
In [16]: my_dict.keys()
```

```
Out[16]: dict_keys(['a', 'b'])
```

```
In [17]: my_dict.items()
```

```
Out[17]: dict_items([('a', 1), ('b', 2)])
```

# Adding and removing from containers

# Adding and removing from containers

In [18]:

```
my_list = [1, 55, 3]
my_dict = {"a": 1, "b": 2}
```

# Adding and removing from containers

In [18]:

```
my_list = [1, 55, 3]
my_dict = {"a": 1, "b": 2}
```

## Adding elements

In [19]:

```
my_list.append(4)
my_dict["c"] = 99
```

# Adding and removing from containers

In [18]:

```
my_list = [1, 55, 3]
my_dict = {"a": 1, "b": 2}
```

## Adding elements

In [19]:

```
my_list.append(4)
my_dict["c"] = 99
```

## Removing elements

In [20]:

```
four = my_list.pop()
my_list.remove(55)
del my_list[0]
print(my_list)
```

```
[3]
```

# Adding and removing from containers

In [18]:

```
my_list = [1, 55, 3]
my_dict = {"a": 1, "b": 2}
```

## Adding elements

In [19]:

```
my_list.append(4)
my_dict["c"] = 99
```

## Removing elements

In [20]:

```
four = my_list.pop()
my_list.remove(55)
del my_list[0]
print(my_list)
```

```
[3]
```

In [21]:

```
one = my_dict.pop("a")
del my_dict["b"]
print(my_dict)
```

```
{'c': 99}
```

# Extending, counting, checking

In [22]:

```
print(my_list)
print(my_dict)
```

```
[3]
{'c': 99}
```

# Extending, counting, checking

In [22]:

```
print(my_list)
print(my_dict)
```

```
[3]
{'c': 99}
```

In [23]:

```
my_list.extend([4, 5, 6, 12, 11, 10])
my_dict.update({"x": -1, "y": -10, "z": -100})
```

# Extending, counting, checking

In [22]:

```
print(my_list)
print(my_dict)
```

```
[3]
{'c': 99}
```

In [23]:

```
my_list.extend([4, 5, 6, 12, 11, 10])
my_dict.update({"x": -1, "y": -10, "z": -100})
```

In [24]:

```
print(f"{len(my_list)} elements: {my_list}")
```

```
7 elements: [3, 4, 5, 6, 12, 11, 10]
```

In [25]:

```
print(f"{len(my_dict)} pairs: {my_dict}")
```

```
4 pairs: {'c': 99, 'x': -1, 'y': -10, 'z': -100}
```

# Extending, counting, checking

In [22]:

```
print(my_list)
print(my_dict)
```

```
[3]
{'c': 99}
```

In [23]:

```
my_list.extend([4, 5, 6, 12, 11, 10])
my_dict.update({"x": -1, "y": -10, "z": -100})
```

In [24]:

```
print(f"{len(my_list)} elements: {my_list}")
```

```
7 elements: [3, 4, 5, 6, 12, 11, 10]
```

In [25]:

```
print(f"{len(my_dict)} pairs: {my_dict}")
```

```
4 pairs: {'c': 99, 'x': -1, 'y': -10, 'z': -100}
```

In [26]:

```
5 in my_list
```

Out[26]:

```
True
```

In [27]:

```
-10 in my_dict
```

Out[27]:

```
False
```

# Sorting

In [28]:

```
print(my_list)
```

```
[3, 4, 5, 6, 12, 11, 10]
```

# Sorting

```
In [28]: print(my_list)

[3, 4, 5, 6, 12, 11, 10]
```

```
In [29]: print(sorted(my_list))

[3, 4, 5, 6, 10, 11, 12]
```

```
In [30]: print(my_list)

[3, 4, 5, 6, 12, 11, 10]
```

# Sorting

```
In [28]: print(my_list)
```

```
[3, 4, 5, 6, 12, 11, 10]
```

```
In [29]: print(sorted(my_list))
```

```
[3, 4, 5, 6, 10, 11, 12]
```

```
In [30]: print(my_list)
```

```
[3, 4, 5, 6, 12, 11, 10]
```

```
In [31]: print(my_list.sort())
```

```
None
```

```
In [32]: print(my_list)
```

```
[3, 4, 5, 6, 10, 11, 12]
```

# Sorting

```
In [28]: print(my_list)
```

```
[3, 4, 5, 6, 12, 11, 10]
```

```
In [29]: print(sorted(my_list))
```

```
[3, 4, 5, 6, 10, 11, 12]
```

```
In [30]: print(my_list)
```

```
[3, 4, 5, 6, 12, 11, 10]
```

```
In [31]: print(my_list.sort())
```

```
None
```

```
In [32]: print(my_list)
```

```
[3, 4, 5, 6, 10, 11, 12]
```

But: No notion of "sorting" a dictionary!

# Other useful containers (1/2)

In [33]:

```
from collections import OrderedDict, Counter
```

# Other useful containers (1/2)

In [33]:

```
from collections import OrderedDict, Counter
```

In [34]:

```
o = OrderedDict()
o["last"] = 9
o["first"] = 2
o["second"] = -1
o.move_to_end("last")
o
```

Out[34]:

```
OrderedDict([('first', 2), ('second', -1), ('last', 9)])
```

# Other useful containers (1/2)

In [33]:

```
from collections import OrderedDict, Counter
```

In [34]:

```
o = OrderedDict()
o["last"] = 9
o["first"] = 2
o["second"] = -1
o.move_to_end("last")
o
```

Out[34]:

```
OrderedDict([('first', 2), ('second', -1), ('last', 9)])
```

In [35]:

```
c = Counter()
c["a"] = 2
c["b"] += 3
c
```

Out[35]:

```
Counter({'a': 2, 'b': 3})
```

# Other useful containers (1/2)

In [33]:

```
from collections import OrderedDict, Counter
```

In [34]:

```
o = OrderedDict()
o["last"] = 9
o["first"] = 2
o["second"] = -1
o.move_to_end("last")
o
```

Out[34]:

```
OrderedDict([('first', 2), ('second', -1), ('last', 9)])
```

In [35]:

```
c = Counter()
c["a"] = 2
c["b"] += 3
c
```

Out[35]:

```
Counter({'a': 2, 'b': 3})
```

In [36]:

```
c.most_common()
```

Out[36]:

```
[('b', 3), ('a', 2)]
```

In [37]:

```
c.total()
```

```
Out[37]:    5
```

# Other useful containers (2/2)

In [38]:

```
from collections import defaultdict
```

# Other useful containers (2/2)

In [38]:

```
from collections import defaultdict
```

In [39]:

```
d = defaultdict(list)
d
```

Out[39]:

```
defaultdict(list, {})
```

# Other useful containers (2/2)

```
In [38]: from collections import defaultdict
```

```
In [39]: d = defaultdict(list)
         d
```

```
Out[39]:   defaultdict(list, {})
```

```
In [40]: d["first_names"].append("Frank")
         d["last_names"].extend(["Zieglmeier", "Zett", "Thelen"])
         d
```

```
Out[40]:   defaultdict(list,
                       {'first_names': ['Frank'],
                        'last_names': ['Zieglmeier', 'Zett', 'Thelen']})
```

# Selection by index and range (slicing)

```
In [41]: d = {"a": 99, "b": 22.0}
         d["b"]

Out[41]: 22.0
```

# Selection by index and range (slicing)

```
In [41]: d = {"a": 99, "b": 22.0}
         d["b"]
```

```
Out[41]: 22.0
```

```
In [42]: l = [1, 2, 3, 4, 5]
         l[0]  # first element
```

```
Out[42]: 1
```

# Selection by index and range (slicing)

```
In [41]: d = {"a": 99, "b": 22.0}
         d["b"]

Out[41]: 22.0
```

```
In [42]: l = [1, 2, 3, 4, 5]
         l[0]  # first element

Out[42]: 1
```

```
In [43]: l[-1]  # last element

Out[43]: 5
```

# Selection by index and range (slicing)

```
In [41]: d = {"a": 99, "b": 22.0}
         d["b"]

Out[41]: 22.0

In [42]: l = [1, 2, 3, 4, 5]
         l[0]  # first element

Out[42]: 1

In [43]: l[-1]  # last element

Out[43]: 5

In [44]: l[1:4]   # select a "slice"

Out[44]: [2, 3, 4]
```

# → Lesson #2

# Conditions

# Conditions

In [45]:

```
number = 1

if number > 0:
    print("positive")
elif number == 0:
    print("zero")
else:
    print("number not allowed")
```

```
positive
```

# Conditions

In [45]:

```
number = 1

if number > 0:
    print("positive")
elif number == 0:
    print("zero")
else:
    print("number not allowed")
```

```
positive
```

In [46]:

```
my_list = []
if not my_list:  # equivalent to `if len(my_list) == 0`
    print("list is empty")
```

```
list is empty
```

# Loops

# Loops

In [47]:

```
l = [1, 2, 3, 4]
for i in l:
    print(i)
```

```
1
2
3
4
```

# Loops

In [47]:

```
l = [1, 2, 3, 4]
for i in l:
    print(i)
```

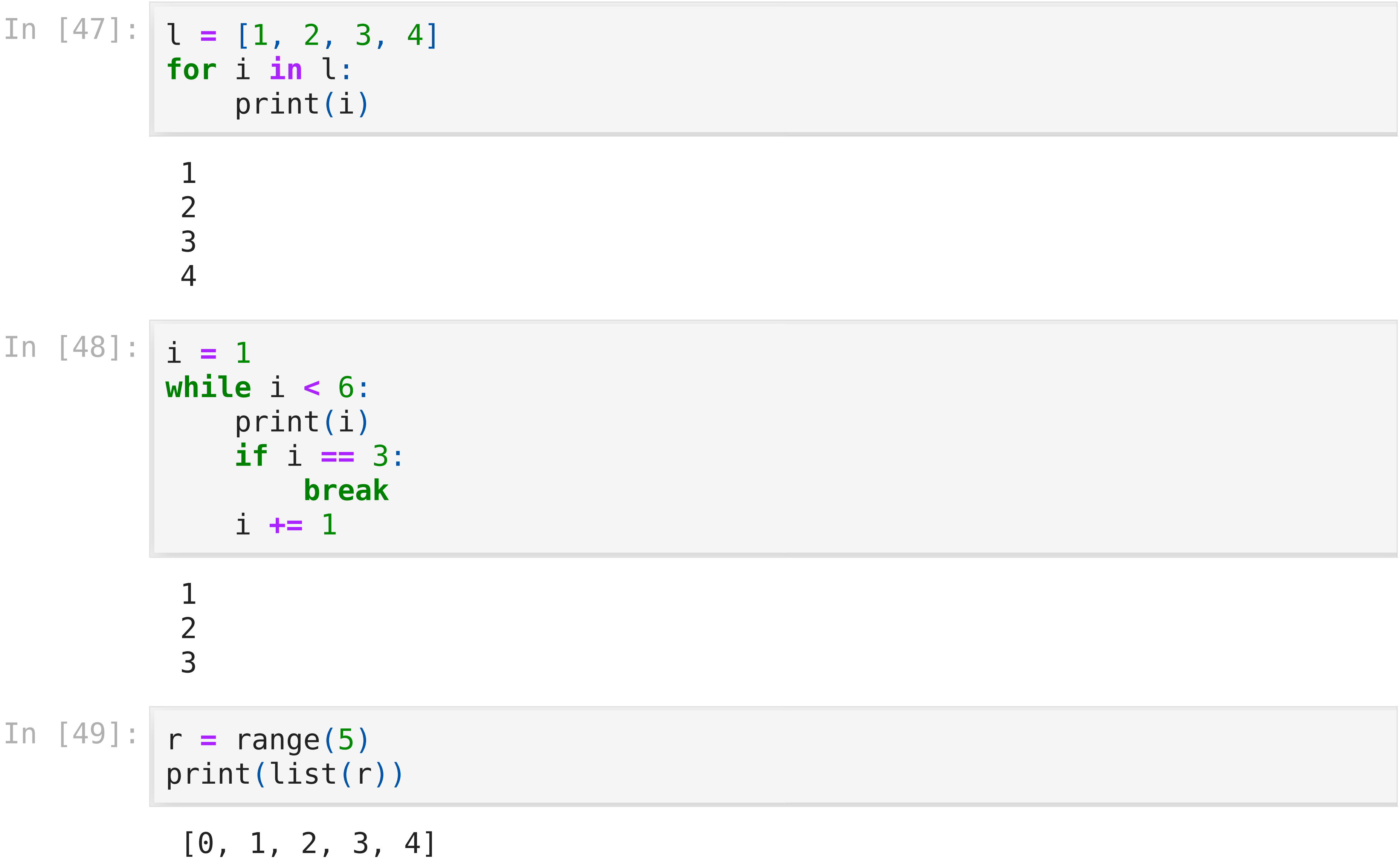
```
1
2
3
4
```

In [48]:

```
i = 1
while i < 6:
    print(i)
    if i == 3:
        break
    i += 1
```

```
1
2
3
```

# Loops



```
In [47]: l = [1, 2, 3, 4]
         for i in l:
             print(i)
```

```
1
2
3
4
```

```
In [48]: i = 1
         while i < 6:
             print(i)
             if i == 3:
                 break
             i += 1
```

```
1
2
3
```

```
In [49]: r = range(5)
         print(list(r))
```

```
[0, 1, 2, 3, 4]
```

# Functions

# Functions

In [50]:

```
def number_add(number_1, number_2):
    return number_1 + number_2
```

# Functions

```
In [50]: def number_add(number_1, number_2):
             return number_1 + number_2
```

```
In [51]: number_add(4, 2)
```

```
Out[51]: 6
```

# Functions

```
In [50]: def number_add(number_1, number_2):
             return number_1 + number_2
```

```
In [51]: number_add(4, 2)
```

```
Out[51]: 6
```

```
In [52]: number_add(5.5, 9.1)   # float supports `+` as well
```

```
Out[52]: 14.6
```

# Functions

```
In [50]: def number_add(number_1, number_2):
             return number_1 + number_2
```

```
In [51]: number_add(4, 2)
```

```
Out[51]: 6
```

```
In [52]: number_add(5.5, 9.1)   # float supports `+` as well
```

```
Out[52]: 14.6
```

## Calling a function from another function

```
In [53]: def process(elements):
             add_to = 10
             added_elements = []
             for element in elements:
                 result = number_add(element, add_to)
                 added_elements.append(result)
             return added_elements

         process([1, 10, 100, 1000])
```

```
Out[53]:    [11, 20, 110, 1010]
```

→ Lessons #3 & #4

# Python: advanced features

# Files

# Files

In [54]:

```
file_name = "foo_bar.txt"
lines = [
    "first line\n",
    "second line\n",
]

with open(file_name, "w") as f:
    f.writelines(lines)
```

# Files

In [54]:

```
file_name = "foo_bar.txt"
lines = [
    "first line\n",
    "second line\n",
]

with open(file_name, "w") as f:
    f.writelines(lines)
```

## content of foo_bar.txt now:

first line
second line

# Files

In [54]:

```
file_name = "foo_bar.txt"
lines = [
    "first line\n",
    "second line\n",
]

with open(file_name, "w") as f:
    f.writelines(lines)
```

## content of foo_bar.txt now:

first line
second line

In [55]:

```
with open(file_name) as f:
    print(f.readlines())
```

```
['first line\n', 'second line\n']
```

In [56]:

```
# re-open the file to start reading again
with open(file_name) as f:
    for line in f:
        print(line.removesuffix("\n"))
```

```
first line
second line
```

# RegEx (1/2)

How to identify and select parts of strings based on their content?

# RegEx (1/2)

How to identify and select parts of strings based on their content?

## Selectors:

| Selector | Meaning | Example |
|---|---|---|
| `<character>` | that character | "a" -> "a", "ab c 1" -> "ab c 1" |
| `[<c><d>...]` | any character in [] | "[ab]c" -> "ac", "bc", but not "ad" |
| `.` | any character | "..." -> "abc", but also "d f" or "1y+" |
| `\w` | any letter | "\w\w\w" -> "abc", "def", but not "d f" |
| `\d` | any digit | "\d" -> "1", but not " " or "a" |
| `\s` | any space | |
| `\W`, `\D`, `\S` | opposite of \w etc. | |

# RegEx (1/2)

How to identify and select parts of strings based on their content?

## Selectors:

| **Selector** | **Meaning** | **Example** |
|---|---|---|
| `<character>` | that character | "a" -> "a", "ab c 1" -> "ab c 1" |
| `[<c><d>...]` | any character in [] | "[ab]c" -> "ac", "bc", but not "ad" |
| `.` | any character | "..." -> "abc", but also "d f" or "1y+" |
| `\w` | any letter | "\w\w\w" -> "abc", "def", but not "d f" |
| `\d` | any digit | "\d" -> "1", but not " " or "a" |
| `\s` | any space | |
| `\W`, `\D`, `\S` | opposite of \w etc. | |

## Quantifiers:

| **Quantifier** | **Meaning** |
|---|---|
| `{<number>}` | `<number>` times |
| `{<n>, <m>}` | between `<n>` and `<m>` times |

| Quantifier | Meaning |
|---|---|
| `+` | at least once |
| `*` | 0 times or more |
| `?` | 0 times or once |

# RegEx (2/2)

Applying it in Python

# RegEx (2/2)

Applying it in Python

In [57]:

```
import re
source = "We can select words, 50, xyz, 20, abc, ?!, ..."
```

# RegEx (2/2)

Applying it in Python

In [57]:

```
import re
source = "We can select words, 50, xyz, 20, abc, ?!, ..."
```

In [58]:

```
match = re.search("words", source)
if match:
    print(f"found it! at: {match.span()}")
```

```
found it! at: (14, 19)
```

# RegEx (2/2)

Applying it in Python

In [57]:

```
import re
source = "We can select words, 50, xyz, 20, abc, ?!, ..."
```

In [58]:

```
match = re.search("words", source)
if match:
    print(f"found it! at: {match.span()}")
```

```
found it! at: (14, 19)
```

In [59]:

```
start, end = match.span()
source[start:end]
```

Out[59]:

```
'words'
```

# RegEx (2/2)

Applying it in Python

In [57]:

```
import re
source = "We can select words, 50, xyz, 20, abc, ?!, ..."
```

In [58]:

```
match = re.search("words", source)
if match:
    print(f"found it! at: {match.span()}")
```

```
found it! at: (14, 19)
```

In [59]:

```
start, end = match.span()
source[start:end]
```

Out[59]:

```
'words'
```

In [60]:

```
re.findall(r"\d{2}", source)
```

Out[60]:

```
['50', '20']
```

# RegEx (2/2)

Applying it in Python

```
In [57]: import re
         source = "We can select words, 50, xyz, 20, abc, ?!, ..."
```

```
In [58]: match = re.search("words", source)
         if match:
             print(f"found it! at: {match.span()}")

          found it! at: (14, 19)
```

```
In [59]: start, end = match.span()
         source[start:end]

Out[59]:  'words'
```

```
In [60]: re.findall(r"\d{2}", source)

Out[60]:  ['50', '20']
```

```
In [61]: match = re.search(r".0,+", source)
         match[0]

Out[61]:  '50,'
```

→ Lessons #5 & #6

# Collaborative Coding Challenge

Create a program to manage cooking recipes

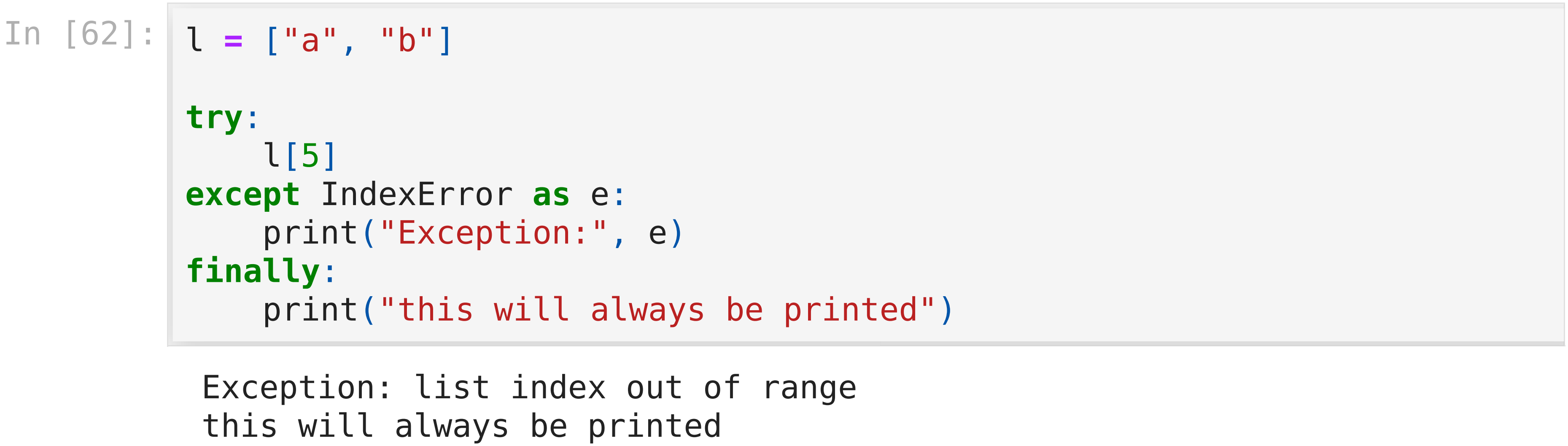
- store recipes
- add & remove recipes
- list all recipes

Advanced features:

- save & load recipes to/from a file
- search by ingredient
- calculate total ingredients needed

# Exception Handling

# Exception Handling



In [62]:

```
l = ["a", "b"]

try:
    l[5]
except IndexError as e:
    print("Exception:", e)
finally:
    print("this will always be printed")
```

```
Exception: list index out of range
this will always be printed
```

# Exception Handling

In [62]:

```
l = ["a", "b"]

try:
    l[5]
except IndexError as e:
    print("Exception:", e)
finally:
    print("this will always be printed")
```

```
Exception: list index out of range
this will always be printed
```

In [63]:

```
raise ValueError("Something went wrong.")
```

```
-----------------------------------------------------------------
-----------
ValueError                                 Traceback (most recent
call last)
Cell In[63], line 1
----> 1 raise ValueError("Something went wrong.")

ValueError: Something went wrong.
```

# Pattern matching (1/2)

Dealing with dynamically shaped data.

# Pattern matching (1/2)

Dealing with dynamically shaped data.

Let's handle dynamic commands:

- `"process"` -> trigger some processing
- `"go up"` -> navigate to parent directory
- `"open results_2025-01"` -> open the directory named "results_2025-01"

# Pattern matching (2/2)

# Pattern matching (2/2)

In [64]:

```
def handle_command(cmd: str):
    match cmd.split():
        case ["process"]:
            print("processing results...")
        case ["go", "up"]:
            print("navigating back up...")  # os.chdir("..")
        case ["open", name]:
            print(f"opening {name}...")  # os.chdir(name)
        case _:
            raise ValueError(f"could not parse command: {cmd}")
```

# Pattern matching (2/2)

In [64]:

```
def handle_command(cmd: str):
    match cmd.split():
        case ["process"]:
            print("processing results...")
        case ["go", "up"]:
            print("navigating back up...")  # os.chdir("..")
        case ["open", name]:
            print(f"opening {name}...")  # os.chdir(name)
        case _:
            raise ValueError(f"could not parse command: {cmd}")
```

In [65]:

```
handle_command("process")
```

```
processing results...
```

In [66]:

```
handle_command("go up")
```

```
navigating back up...
```

In [67]:

```
handle_command("open results_2025-01")
```

```
opening results_2025-01...
```

→ Lessons #7 & #8

# Closing thoughts

# Best practices (1/2)

# Best practices (1/2)

In [68]:

```
def f(p, d):
    dp = p * (1 - d / 100)
    if dp < 50:
        return 50
    else:
        return dp


x = 100
y = 20
z = f(x, y)  # What shall that be?
```

# Best practices (2/2)

In [69]:

```
MINIMUM_PRICE: float = 50.0  # extract constants
MAX_DISCOUNT: float = 80.0

# separate functionality
# use meaningful names
# use type hints (Python 3.5+) and mypy

def apply_minimum_price(price: float) -> float:
    """Ensure price never falls below minimum"""  # write comments
    return max(price, MINIMUM_PRICE)

def calculate_discounted_price(
    original: float,
    discount_pct: float,
) -> float:
    """
    Calculate final price after discount.
    Applies minimum price rule automatically.
    """
    if discount_pct > MAX_DISCOUNT:  # input validation
        raise ValueError("Discount exceeds maximum")

    discounted = original * (1 - discount_pct / 100)  # use proper white
    return apply_minimum_price(discounted)

item_price = 100.0
```

```
discount = 20.0
final_price = calculate_discounted_price(item_price, discount)
```

# How to start my own project

# How to start my own project

```
uv init my_project
```

# How to start my own project

```
uv init my_project
```

```
cd my_project
uv add peek-python  # adding a dependency
code hello.py
```

# How to start my own project

```
uv init my_project
```

```
cd my_project
uv add peek-python  # adding a dependency
code hello.py
```

## content of hello.py

```
from peek import peek

def add2(i):
    peek()          # where are we right now?
    return i + 2

class X:
    a = 3

world = {"EN": "world", "NL": "wereld"}
peek(add2(1000))  # what is the result of this?
peek(world, X.a)  # what is the value of these?
```

# How to start my own project

```
uv init my_project
```

```
cd my_project
uv add peek-python  # adding a dependency
code hello.py
```

## content of hello.py

```
from peek import peek

def add2(i):
    peek()          # where are we right now?
    return i + 2

class X:
    a = 3

world = {"EN": "world", "NL": "wereld"}
peek(add2(1000))  # what is the result of this?
peek(world, X.a)  # what is the value of these?
```

## running our code

```
> uv run hello.py
```

# How to start my own project

```
uv init my_project
```

```
cd my_project
uv add peek-python  # adding a dependency
code hello.py
```

## content of hello.py

```
from peek import peek

def add2(i):
    peek()          # where are we right now?
    return i + 2

class X:
    a = 3

world = {"EN": "world", "NL": "wereld"}
peek(add2(1000))  # what is the result of this?
peek(world, X.a)  # what is the value of these?
```

## running our code

```
> uv run hello.py
```

```
#4 in add2()
add2(1000)=1002
X.a=3, world={'EN': 'world', 'NL': 'wereld'}
```

# Backup

# Operators

# Operators

In [70]:

```
print(5 ** 2)  # exponentiation
print(5 // 2)  # floor division
print(10 % 7)  # modulo
```

```
25
2
3
```

# Functions

# Functions

In [71]:

```
def str_add(x, prefix='Argument: '):
    return prefix + str(x)
```

# Functions

```
In [71]: def str_add(x, prefix='Argument: '):
             return prefix + str(x)
```

```
In [72]: str_add(4)   # positional argument
```

```
Out[72]: 'Argument: 4'
```

# Functions

```
In [71]: def str_add(x, prefix='Argument: '):
             return prefix + str(x)
```

```
In [72]: str_add(4)   # positional argument

Out[72]: 'Argument: 4'
```

```
In [73]: str_add(4, 'P')

Out[73]: 'P4'
```

# Functions

In [71]:

```
def str_add(x, prefix='Argument: '):
    return prefix + str(x)
```

In [72]:

```
str_add(4)   # positional argument
```

Out[72]:

```
'Argument: 4'
```

In [73]:

```
str_add(4, 'P')
```

Out[73]:

```
'P4'
```

In [74]:

```
# alternatives using keyword arguments
str_add(4, prefix='P')
str_add(x=4, prefix='P')
str_add(prefix='P', x=4)
str_add(x=4, prefix='P')
```

Out[74]:

```
'P4'
```

# Functions

```
In [71]: def str_add(x, prefix='Argument: '):
             return prefix + str(x)
```

```
In [72]: str_add(4)   # positional argument

Out[72]:   'Argument: 4'
```

```
In [73]: str_add(4, 'P')

Out[73]:   'P4'
```

```
In [74]: # alternatives using keyword arguments
         str_add(4, prefix='P')
         str_add(x=4, prefix='P')
         str_add(prefix='P', x=4)
         str_add(x=4, prefix='P')

Out[74]:   'P4'
```

```
In [75]: new_str_add = str_add   # functions are first class citizen
         new_str_add(4)

Out[75]:   'Argument: 4'
```

# Functions

In [71]:

```
def str_add(x, prefix='Argument: '):
    return prefix + str(x)
```

In [72]:

```
str_add(4)   # positional argument
```

Out[72]:

```
'Argument: 4'
```

In [73]:

```
str_add(4, 'P')
```

Out[73]:

```
'P4'
```

In [74]:

```
# alternatives using keyword arguments
str_add(4, prefix='P')
str_add(x=4, prefix='P')
str_add(prefix='P', x=4)
str_add(x=4, prefix='P')
```

Out[74]:

```
'P4'
```

In [75]:

```
new_str_add = str_add   # functions are first class citizen
new_str_add(4)
```

Out[75]:

```
'Argument: 4'
```

```
In [76]: (lambda x: x + 5)(4)    # anonymous function

Out[76]: 9
```

# Comprehensions

# Comprehensions

```
In [77]: [i**2 for i in range(10) if i % 2 == 0]    # list comprehension
```

```
Out[77]: [0, 4, 16, 36, 64]
```

# Comprehensions

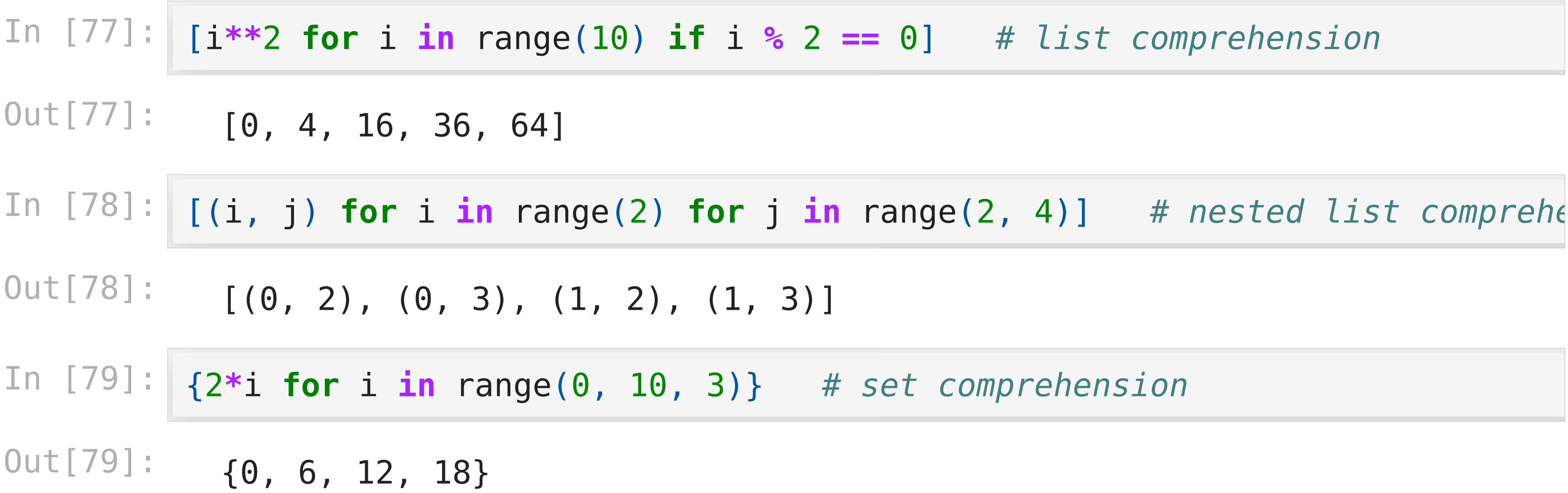
```
In [77]: [i**2 for i in range(10) if i % 2 == 0]   # list comprehension
```

```
Out[77]: [0, 4, 16, 36, 64]
```

```
In [78]: [(i, j) for i in range(2) for j in range(2, 4)]   # nested list comprehe
```

```
Out[78]: [(0, 2), (0, 3), (1, 2), (1, 3)]
```

# Comprehensions



```
In [77]: [i**2 for i in range(10) if i % 2 == 0]    # list comprehension

Out[77]: [0, 4, 16, 36, 64]

In [78]: [(i, j) for i in range(2) for j in range(2, 4)]    # nested list comprehe

Out[78]: [(0, 2), (0, 3), (1, 2), (1, 3)]

In [79]: {2*i for i in range(0, 10, 3)}    # set comprehension

Out[79]: {0, 6, 12, 18}
```

# Comprehensions

```
In [77]: [i**2 for i in range(10) if i % 2 == 0]   # list comprehension

Out[77]:  [0, 4, 16, 36, 64]

In [78]: [(i, j) for i in range(2) for j in range(2, 4)]   # nested list comprehe

Out[78]:  [(0, 2), (0, 3), (1, 2), (1, 3)]

In [79]: {2*i for i in range(0, 10, 3)}   # set comprehension

Out[79]:  {0, 6, 12, 18}

In [80]: {i: i ** 2 for i in range(5, 0, -1)}   # dict comprehension

Out[80]:  {5: 25, 4: 16, 3: 9, 2: 4, 1: 1}
```

# * and ** Operators

In [81]:

```
def f(a, b):
    print(a, b)
```

# * and ** Operators

In [81]:

```
def f(a, b):
    print(a, b)
```

In [82]:

```
# unpack positional arguments
l = [3, 6]
f(*l)
```

```
3 6
```

# * and ** Operators

In [81]:
```
def f(a, b):
    print(a, b)
```

In [82]:
```
# unpack positional arguments
l = [3, 6]
f(*l)
```

```
3 6
```

In [83]:
```
# unpack keyword arguments
d = {'b': 6, 'a': 3}
f(**d)
```

```
3 6
```

# * and ** Operators

In [81]:

```
def f(a, b):
    print(a, b)
```

In [82]:

```
# unpack positional arguments
l = [3, 6]
f(*l)
```

```
3 6
```

In [83]:

```
# unpack keyword arguments
d = {'b': 6, 'a': 3}
f(**d)
```

```
3 6
```

In [84]:

```
# functions with arbitrary number of arguments

def fun_with_args_and_kwargs(a, b=3, *args, **kwargs):
    print(f'a={a}, b={b}, args={args}, kwargs={kwargs}')

fun_with_args_and_kwargs(1, 2)
fun_with_args_and_kwargs(1, 2, 3)
fun_with_args_and_kwargs(1, c=7)
fun_with_args_and_kwargs(1, b= 5, c=7)
```

```
a=1, b=2, args=(), kwargs={}
a=1, b=2, args=(3,), kwargs={}
a=1, b=3, args=(), kwargs={'c': 7}
a=1, b=5, args=(), kwargs={'c': 7}
```

# Classes

# Classes

In [85]:

```
class Rectangle:
    def __init__(self, length, width):
        self._length = length
        self._width = width

    def get_area(self):
        return self._length * self._width

r = Rectangle(2, 3)
```

# Classes

In [85]:

```
class Rectangle:
    def __init__(self, length, width):
        self._length = length
        self._width = width

    def get_area(self):
        return self._length * self._width

r = Rectangle(2, 3)
```

In [86]:

```
type(Rectangle), type(r)
```

Out[86]:

```
(type, __main__.Rectangle)
```

# Classes

In [85]:

```
class Rectangle:
    def __init__(self, length, width):
        self._length = length
        self._width = width

    def get_area(self):
        return self._length * self._width

r = Rectangle(2, 3)
```

In [86]:

```
type(Rectangle), type(r)
```

Out[86]:

```
(type, __main__.Rectangle)
```

In [87]:

```
r.get_area(), r._length, r._width
```

Out[87]:

```
(6, 2, 3)
```

# Magic Methods

# Magic Methods

In [88]:

```python
class MyClass:
    def __len__(self):
        return 2

c = MyClass()
len(c)
```

Out[88]:

```
2
```

# Iterators

# Iterators

In [89]:

```
class MyRange:
    def __init__(self, n):
        self.i = 0
        self.n = n
    def __iter__(self):
        return self
    def __next__(self):
        if self.i < self.n:
            i = self.i
            self.i += 1
            return i
        else:
            raise StopIteration()
```

# Iterators

In [89]:

```
class MyRange:
    def __init__(self, n):
        self.i = 0
        self.n = n
    def __iter__(self):
        return self
    def __next__(self):
        if self.i < self.n:
            i = self.i
            self.i += 1
            return i
        else:
            raise StopIteration()
```

In [90]:

```
for i in MyRange(2):
    print(i)
```

```
0
1
```

# Iterators

In [89]:

```
class MyRange:
    def __init__(self, n):
        self.i = 0
        self.n = n
    def __iter__(self):
        return self
    def __next__(self):
        if self.i < self.n:
            i = self.i
            self.i += 1
            return i
        else:
            raise StopIteration()
```

In [90]:

```
for i in MyRange(2):
    print(i)
```

```
0
1
```

In [91]:

```
iterable = MyRange(2)
iterator = iter(iterable)
print(next(iterator))
print(next(iterator))
print(next(iterator))   # StopIteration exception thrown
```

```
0
1
```

```
-----------------------------------------------------------------
-----------
StopIteration                           Traceback (most recent
call last)
Cell In[91], line 5
      3 print(next(iterator))
      4 print(next(iterator))
----> 5 print(next(iterator))   # StopIteration exception thrown

Cell In[89], line 13, in MyRange.__next__(self)
     11     return i
     12 else:
---> 13     raise StopIteration()

StopIteration:
```

# Generators

# Generators

In [92]:

```
def my_range(n):
    i = 0
    while i < n:
        yield i
        i += 1
```

# Generators

In [92]:

```
def my_range(n):
    i = 0
    while i < n:
        yield i
        i += 1
```

In [93]:

```
for i in my_range(3):
    print(i)
```

```
0
1
2
```

In [94]:

```
plus_range = (i+1 for i in my_range(3))
```

In [95]:

```
for i in plus_range:
    print(i)
```

```
1
2
3
```

# Loops - Advanced

# Loops - Advanced

In [96]:

```
for i in l:
    print(i)
else:
    print('else')
```

```
3
6
else
```

In [97]:

```
## Files - without Context Manager
```

# Loops - Advanced

In [96]:

```
for i in l:
    print(i)
else:
    print('else')
```

```
3
6
else
```

In [97]:

```
## Files - without Context Manager
```

In [98]:

```
f = open('foo_bar.txt', 'r')
try:
    for line in f:
        print(line)
finally:
    f.close()
```

```
first line

second line
```