The Python Language

Hendrik Speleers

The Python Language

Università di Roma Tor Vergata

- Overview
  - IPython/Jupyter
  - Data types
    - Numerics, tuples, lists and strings
  - Control flow
    - Selection: if, if-elif-else
    - Repetition: while, for
  - Reusing code
    - Functions
    - Scripts and modules

The Python Language

Università di Roma Tor Vergata

- IPython/Jupyter: tips and tricks
  - Getting help by using ? after an object
  - Tab completion to view object's attributes (object.<TAB>)
  - Magic functions, prefixed with % sign
    - `%run` : execute python script
    - `%whos` : overview variables
    - `%cd` : change the current working directory
    - `%timeit` : time the execution of short snippits
    - `%quickref` : show a cheat-sheet
    - `%automagic` : omit the % sign (default on)

The Python Language

Università di Roma Tor Vergata

- Special characters for instructions
  - Use # to indicate comments
  - Use ;
    - to suppress the printing of output
    - to separate multiple statements on the same line (discouraged)

```
In [1]: 1 + 1                  # output
Out[1]: 2
In [2]: 1 + 1;               # no output
In [3]: print(1 + 1);        # output
2
```

The Python Language

Università di Roma Tor Vergata

- Data types: python example
  - No need to declare the type of variables

```
In [1]: a = 3              # integer
In [2]: type(a)
Out[2]: int
In [3]: 2 * a
Out[3]: 6
In [4]: a = 'hello'        # string
In [5]: type(a)
Out[5]: str
In [6]: 2 * a
Out[6]: 'hellohello'
```

The Python Language

Università di Roma Tor Vergata

- Data types: numerics
  - Integer

```
In [1]: a = 1 + 1
   ...: type(a)
Out[1]: int
```

  - Float

```
In [1]: a = 2.1
   ...: type(a)
Out[1]: float
```

  - Complex

```
In [1]: a = 1.0 + 1.0j
   ...: type(a)
Out[1]: complex
In [2]: a.real + a.imag
Out[2]: 2.0
```

  - Boolean

```
In [1]: a = False
   ...: type(a)
Out[1]: bool
In [2]: 3 < 4
Out[2]: True
```

The Python Language

Università di Roma Tor Vergata

- Data types: numerics
  - Basic arithmetic operations
    - Addition (+), subtraction (−), multiplication (*), division (/)
    - Power (**), integer division (//), modulo (%)
    - Self-operators (+=, −=, ...)
  - Python2 vs. Python3

```
In [1]: 3 / 2
Out[1]: 1
In [2]: 3 / 2.0
Out[2]: 1.5
```

2

```
In [1]: 3 / 2
Out[1]: 1.5
In [2]: 3 // 2.0
Out[2]: 1.0
```

3

The Python Language

Università di Roma Tor Vergata

- Data types: numerics
  - Type casting: conversion of one type into another
    - Explicit via type cast operator
    - Implicit via promotion to larger type in operations

```
In [1]: float(1)
Out[1]: 1.0
In [2]: 1 + 2.5
Out[2]: 3.5
In [3]: 1 + int(2.5)
Out[3]: 3
```

The Python Language

Università di Roma Tor Vergata

- Data types: tuples
  - A tuple is an ordered, immutable collection of objects
    - Objects may have different types

```
In [1]: tup1 = 1, 2, 3
In [2]: tup2 = ('a', 'b', 'c')
In [3]: tup3 = (1, 2, 'z')
In [4]: type(tup3), len(tup3)
Out[4]: (tuple, 3)
In [5]: x, y = 10, 20          # tuple unpacking
```

  - Data extraction is the same as for lists (see next)

The Python Language

Università di Roma Tor Vergata

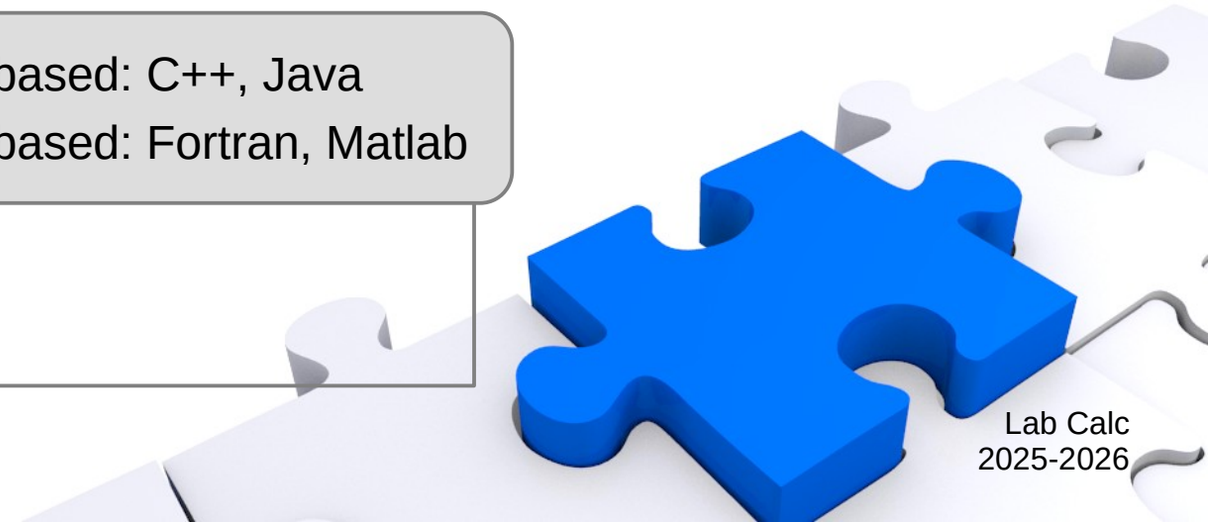
- Data types: lists
  - A list is an ordered collection of objects (may have different types)

```
In [1]: colors = ['red', 'green', 'blue']
   ...: type(colors), len(colors)
Out[1]: (list, 3)
```

  - Indexing: zero-based

```
In [2]: colors[0], colors[1]
Out[2]: ('red', 'green')
In [3]: colors[-1]
Out[3]: 'blue'
```

0-based: C++, Java
1-based: Fortran, Matlab



The Python Language

Università di Roma Tor Vergata

- Data types: lists
  - Slicing syntax: list[start:stop:step]

```
In [1]: colors = ['c', 'o', 'l', 0, 'r', 's']
In [2]: colors[2:4:2]
Out[2]: ['l']
In [3]: colors[2:4]
Out[3]: ['l', 0]
In [4]: colors[:3]
Out[4]: ['c', 'o', 'l']
In [5]: colors[::2]
Out[5]: ['c', 'l', 'r']
```

default:
start=0
stop=len
step=1

The Python Language

Università di Roma Tor Vergata

- Data types: lists
  - Modification of objects

```
In [6]: colors[3] = 'o'; colors
Out[6]: ['c', 'o', 'l', 'o', 'r', 's']
In [7]: colors[:3:2] = ['C', 'L']; colors
Out[7]: ['C', 'o', 'L', 'o', 'r', 's']
```

  - Reverse order

```
In [8]: colors[::-1]        # make new list
Out[8]: ['s', 'r', 'o', 'L', 'o', 'C']
In [9]: colors.reverse()    # update this list
```

The Python Language

Università di Roma Tor Vergata

- Data types: lists
  - Sort list

```
In [8]: sorted(colors)      # make new list
Out[8]: ['C', 'L', 'o', 'o', 'r', 's']
In [9]: colors.sort()       # update this list
```

  - Other methods for list manipulation: list.method
    - `append(object), extend(list)`
    - `insert(index, object)`
    - `pop(), pop(index)`
    - `remove(value)`

example of object-oriented programming

The Python Language

Università di Roma Tor Vergata

- Data types: lists
  - Assignment: pass by reference

```
In [1]: a = [1, 2, 3]
In [2]: b = a; b[1] = 'hi'
In [3]: a
Out[3]: [1, 'hi', 3]
```

  - Indexing/slicing: pass by value

```
In [4]: b[:] = ['a', 'b', 'c']
In [5]: a
Out[5]: ['a', 'b', 'c']
```

to ensure a copy of a list:
b = list(a)
b = a.copy()

The Python Language

Università di Roma Tor Vergata

- Data types: lists
  - Concatenation (+) and repetition (*)

```
In [1]: colors = ['r', 'g', 'b']
In [2]: colors + sorted(colors)
Out[2]: ['r', 'g', 'b', 'b', 'g', 'r']
In [3]: 2 * colors
Out[3]: ['r', 'g', 'b', 'r', 'g', 'b']
```

  - A list can contain other lists
  - For collections of numercial data of same type, more efficient to use `array` from the module `numpy`

The Python Language

Università di Roma Tor Vergata

- Data types: strings
  - Different syntax of quoting

```
s = 'Hi, how are you?'                    # single quotes
s = "What's up, man?"                     # double quotes
s = '''Hello,
   how is it going?'''                    # tripling quotes
s = 'What\'s up, man?'
s = 'Hello,\n   how is it going?'         # using backslash
```

  - Concatenation (+) and repetition (*)

```
s = 2 * 'hello, ' + 'super!'
```

The Python Language

Università di Roma Tor Vergata

- Data types: strings
  - Strings are similar to tuples (immutable lists)

```
In [1]: s = 'Hello, world!'
In [2]: s[3:6], s[-1]
Out[2]: ('lo,', '!')
```

  - Methods for string manipulation: string.method
    - `capitalize(), title(), lower(), upper()`
    - `find(value), replace(old, new)`
    - `partition(sep), split()`
    - `strip(), zfill(width)`

makes a new string (differs from list!)

The Python Language

Università di Roma Tor Vergata

- Data types: strings
  - Formatting

```
In [1]: value = 99
In [2]: 'filename' + str(value) + '.ext'
Out[2]: 'filename99.ext'
In [3]: 'filename%d.ext' % value
Out[3]: 'filename99.ext'
```

  - Formatting placeholders
    - `%s`, `%d`, `%f` : string, integer, float
    - `%.<number of digits>f`
    - Use tuple for multiple formats

The Python Language

- Data types: other container types
  - Dictionary: maps keys to values, unordered

```
In [1]: batch = {'a':1, 'b':2, 'c':3}
   ...: batch['a'] = 5
   ...: batch.keys()
Out[1]: dict_keys(['a', 'b', 'c'])
```

  - Set: unique items, unordered

```
In [1]: batch = {'a', 'b', 'c'}
   ...: batch.difference(['b', 'c'])
Out[1]: {'a'}
```

conversion:
l = list({1, 2})
s = set([1, 2])

The Python Language

Università di Roma Tor Vergata

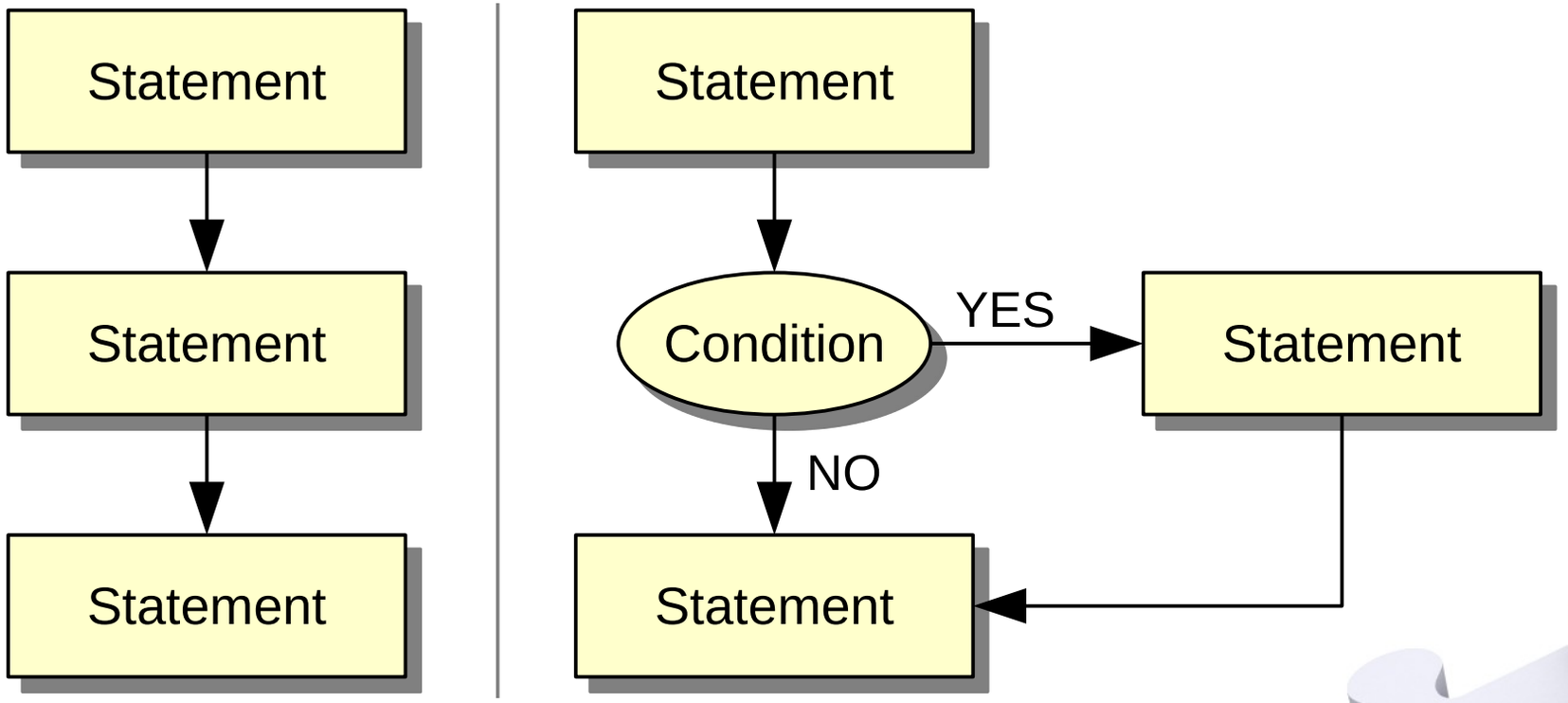
- Control flow
  - Logical operators: `and`, `or`, `not`
  - Conditional expressions
    - `a == b` : tests equality, with logics (similar for `!=`, `<`, `<=`, `>`, `>=`)
    - `a is b` : tests identity, both sides are same object
    - `a in b` : for any collection b: b contains a

a	b	a and b	a or b	not a
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False

```
In [1]: 5 == (3 + 2) and 3 > 0
Out[1]: True
In [2]: 5 not in [1, 2, 3]
Out[2]: True
```

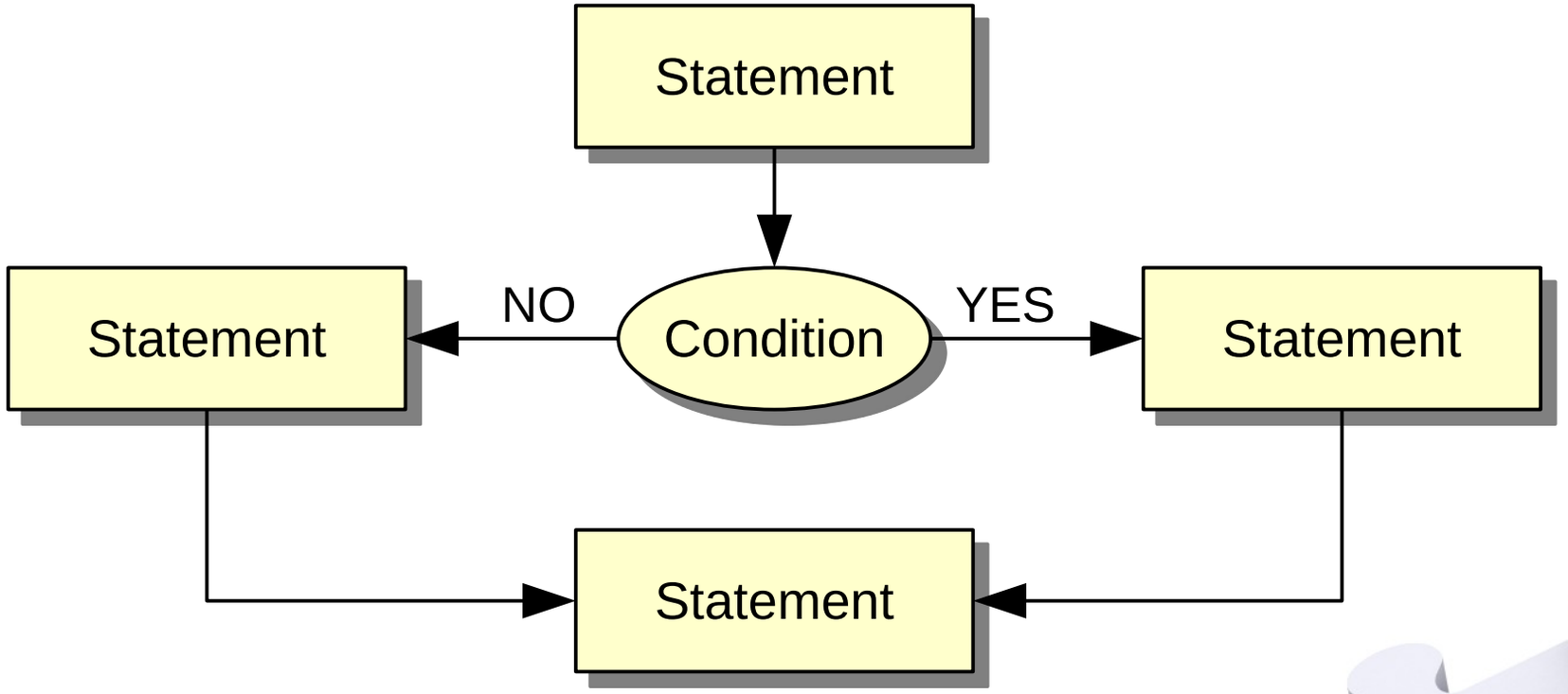
The Python Language

- Control flow: selection



The Python Language

- Control flow: selection



The Python Language

Università di Roma Tor Vergata

- Control flow: selection
  - If statement

```
if condition: statement
if condition:
    block
```

```
In [1]: if 2**2 == 4:
   ...:     print('obvious')
obvious
```

  - Blocks are delimited by indentation (four spaces)
    - No explicit marking of end of block
  - Conditions evaluate to true for
    - boolean `True`
    - any number different from zero (`0, 0.0, 0+0j`)
    - a non-empty collection (tuple, list, ...)

The Python Language

Università di Roma Tor Vergata

- Control flow: selection
  - If-elif-else statement

```
In [1]: a = 10
In [2]: if a == 0:
   ...:     print('zero')
   ...: elif a == 1:
   ...:     print('one')
   ...: elif a == 2:
   ...:     print('two')
   ...: else:
   ...:     print('a lot')
a lot
```

The Python Language

Università di Roma Tor Vergata

- Control flow: selection
  - If-else expression

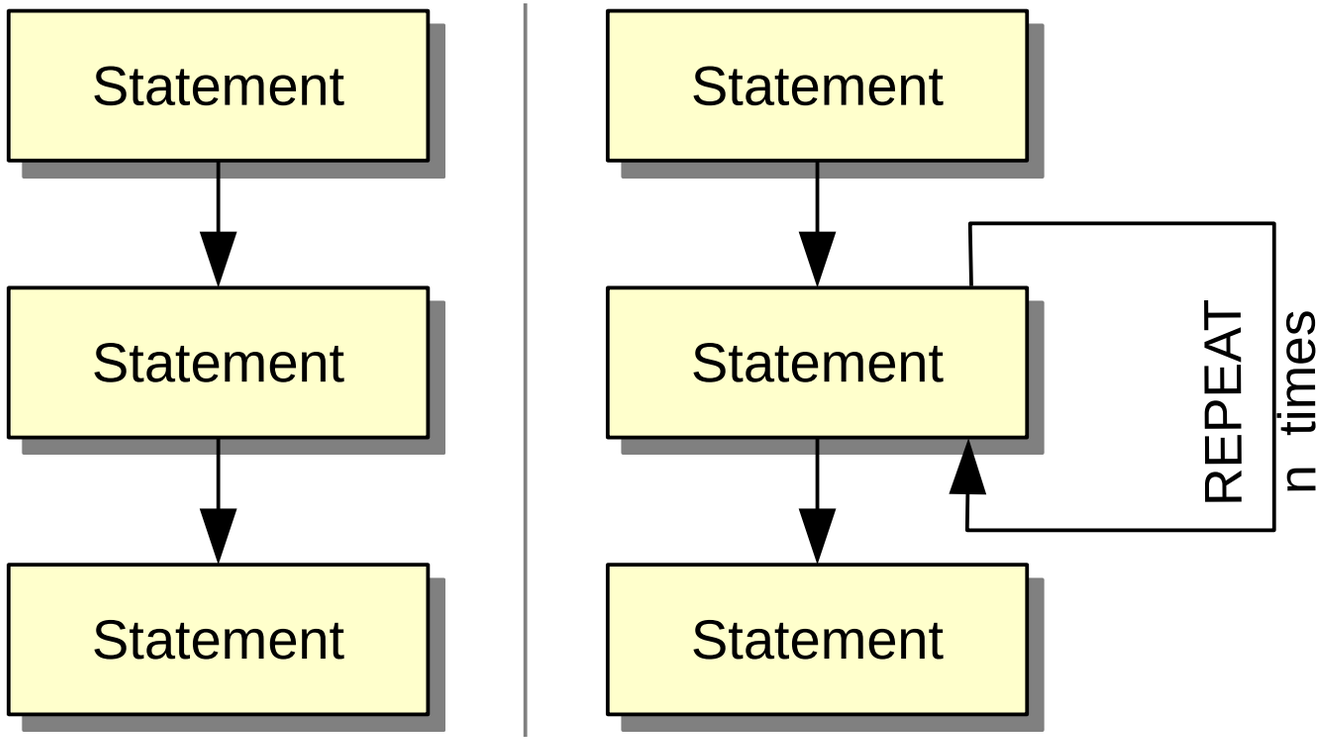
```
expr_true if condition else expr_false
```

```
In [1]: a, b = 10, 20
In [2]: print(a if a > b else b)
20
In [3]: if a > b: print(a)
   ...: else: print(b)
20
```

The Python Language

Università di Roma Tor Vergata

- Control flow: repetition



The Python Language

- Control flow: repetition
  - While loop

```
while condition:
    block
```

  - For loop

```
for item in collection:
    block
```

```
while condition:
    statement
    break
    statement
    continue
    statement
statement
```

  - Altering behavior by `break` and `continue`

The Python Language

- Control flow: repetition
  - While loop

```
while condition:
    block
```

  - For loop

```
for item in collection:
    block
```

```
In [1]: colors = ['r', 'g', 'b']
In [2]: i = 0
   ...: while i < len(colors):
   ...:     print(colors[i])
   ...:     i = i + 1
r
g
b
```

  - Altering behavior by `break` and `continue`

The Python Language

Università di Roma Tor Vergata

- Control flow: repetition
  - While loop

```
while condition:
    block
```

  - For loop

```
for item in collection:
    block
```

```
In [1]: colors = ['r', 'g', 'b']
In [2]: for c in colors:
   ...:     print(c)
r
g
b
```

  - Altering behavior by `break` and `continue`

The Python Language

Università di Roma Tor Vergata

- Control flow: repetition
  - While/for-else loop

```
In [1]: words = ['spam', 'ham', 'eggs']
In [2]: s = 'foo'
In [3]: for w in words:
   ...:     if w == s:
   ...:         # Processing for item found
   ...:         break
   ...: else:
   ...:     # Processing for item not found
   ...:     print(s, 'not found in list')
foo not found in list
```

position is important!

The Python Language

Università di Roma Tor Vergata

- Control flow: repetition
  - Numeric range: `range(stop)` or `range(start, stop, step)`

```
In [1]: numbers = range(5)
In [2]: type(numbers), list(numbers)
Out[2]: (range, [0, 1, 2, 3, 4])
In [3]: for n in numbers:
   ...:     print(n)
0
1
2
3
4
```

The Python Language

- Control flow: repetition
  - Enumeration number: `enumerate(collection, start)`

```
In [1]: words = ['spam', 'ham', 'eggs']
In [2]: for i in range(len(words)):
   ...:     print(i, words[i])
0 spam
1 ham
2 eggs
In [3]: for i, item in enumerate(words):
   ...:     print(i, item)
0 spam
1 ham
2 eggs
```

The Python Language

Università di Roma Tor Vergata

- Control flow: repetition
  - List comprehension

```
[expr for item in collection if condition]
```

```
In [1]: numbers = [4, 3, 1, 5]
In [2]: [n**2 for n in numbers]
Out[2]: [16, 9, 1, 25]
In [3]: [n**2 for n in numbers if n > 2]
Out[3]: [16, 9, 25]
```

The Python Language

Università di Roma Tor Vergata

- Reusing code: functions
  - Defining functions

```
def funname(arguments):
    block
```

  - Return data

```
def funname(arguments):
    block
    return expr
```

```
In [1]: def print_hello():
   ...:     print('hello')
In [2]: print_hello()
hello
```

  - Any number of arguments, possibly optional

The Python Language

Università di Roma Tor Vergata

- Reusing code: functions
  - Defining functions

```
def funname(arguments):
    block
```

  - Return data

```
def funname(arguments):
    block
    return expr
```

```
In [1]: def double_it(x=1):
   ...:     x2 = x * 2
   ...:     return x2
In [2]: double_it(3)
Out[2]: 6
In [3]: double_it()
Out[3]: 2
```

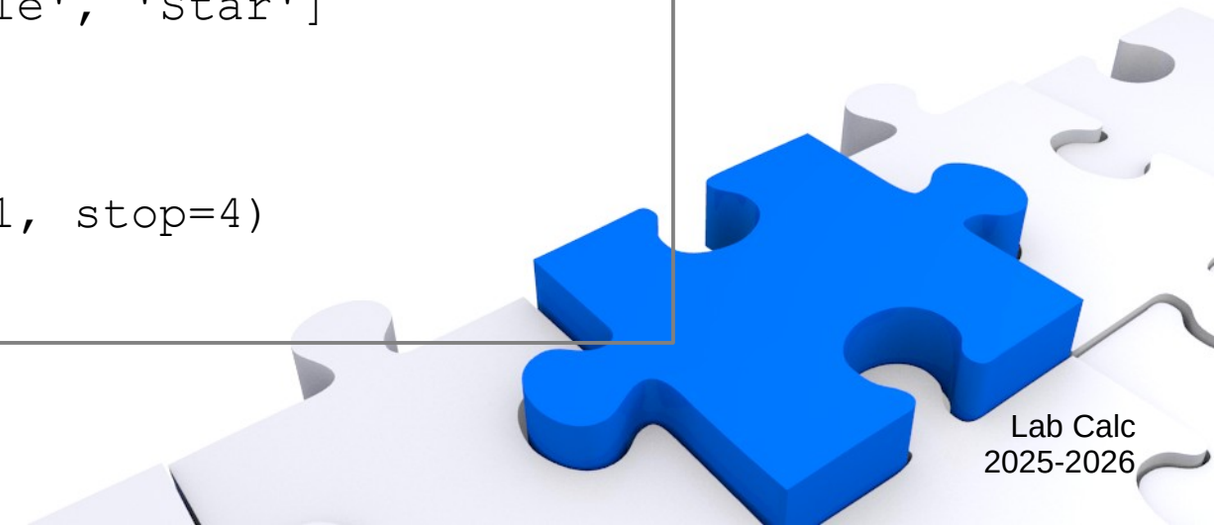
  - Any number of arguments, possibly optional

The Python Language

Università di Roma Tor Vergata

- Reusing code: functions
  - Advanced example of arguments (keywords)

```
In [1]: def slicer(lst, start=None, stop=None, step=None):
   ...:     return lst[start:stop:step]
In [2]: rhyme = 'Twinkle Twinkle Little Star'.split()
In [3]: slicer(rhyme)
Out[3]: ['Twinkle', 'Twinkle', 'Little', 'Star']
In [4]: slicer(rhyme, 1, step=2)
Out[4]: ['Twinkle', 'Star']
In [5]: slicer(rhyme, step=2, start=1, stop=4)
Out[5]: ['Twinkle', 'Star']
```

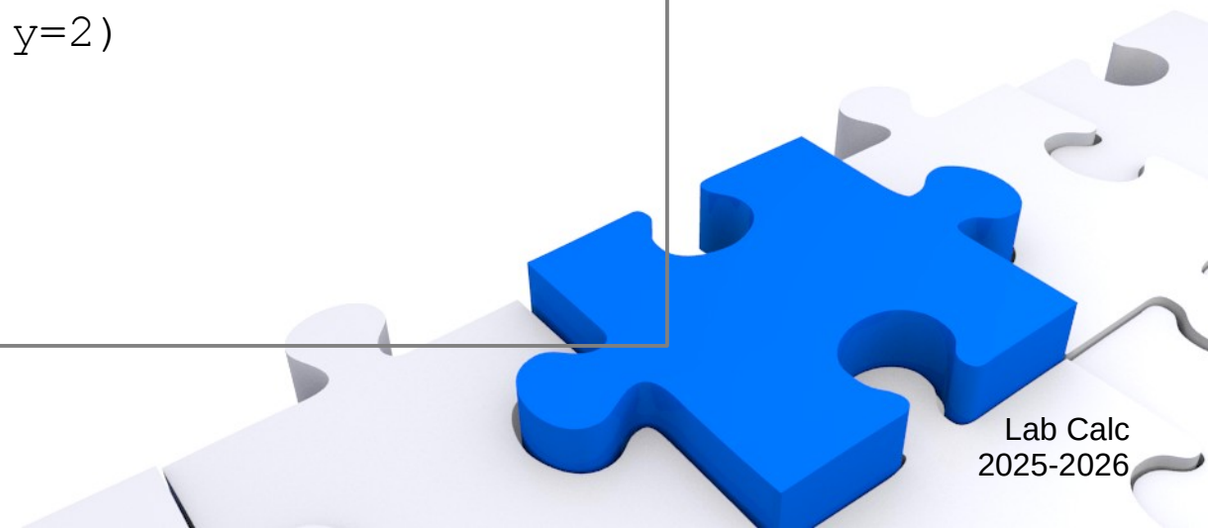


The Python Language

Università di Roma Tor Vergata

- Reusing code: functions
  - Varying number of arguments (* packed into tuple and ** into dictionary)

```
In [1]: def print_all(*args, **kwargs):
   ...:     for arg in args:
   ...:         print(arg)
   ...:     for key, arg in kwargs.items():
   ...:         print((key, arg))
In [2]: print_all('one', 'two', x=1, y=2)
one
two
('x', 1)
('y', 2)
```

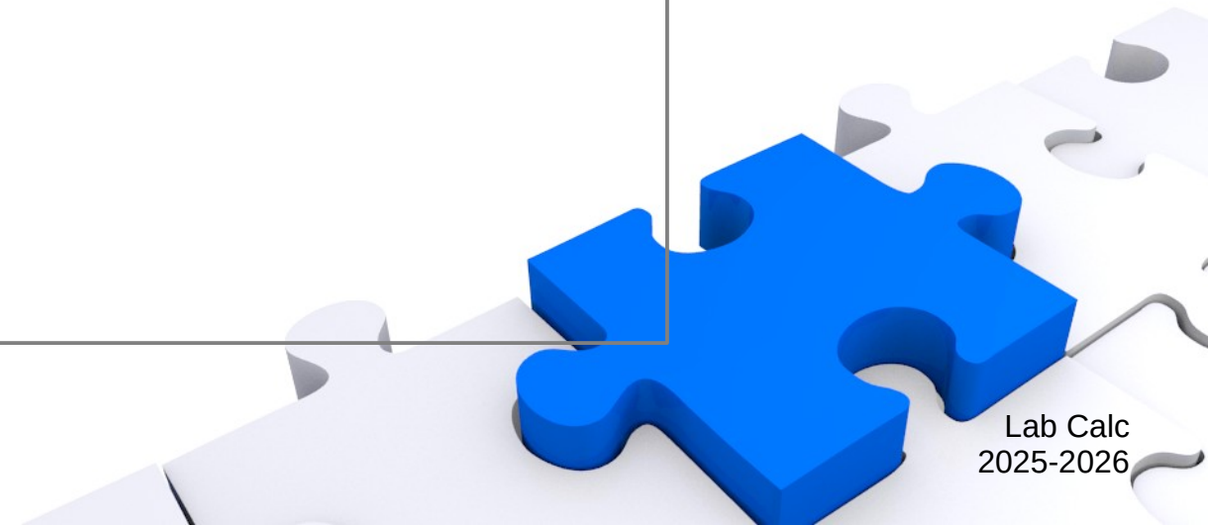


The Python Language

Università di Roma Tor Vergata

- Reusing code: functions
  - Passing of variables: use reference whenever possible
    - immutable (numerics, strings, tuples) vs. mutable (lists)

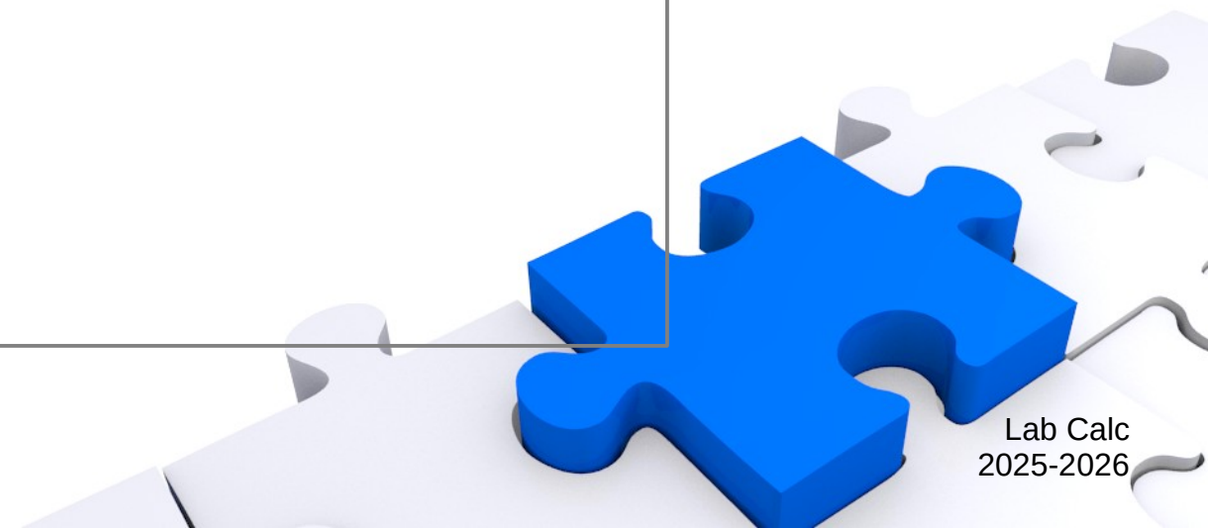
```
In [1]: def try_to_modify(x, y, z):
   ...:     x = 23; y.append(42); z = [99]
   ...:     print(x, y, z)
In [2]: a = 77; b = [9]; c = [28]
In [3]: try_to_modify(a, b, c)
23 [9, 42] [99]
In [4]: print(a, b, c)
77 [9, 42] [28]
```



The Python Language

- Reusing code: functions
  - Local vs. global variables: scope and lifetime

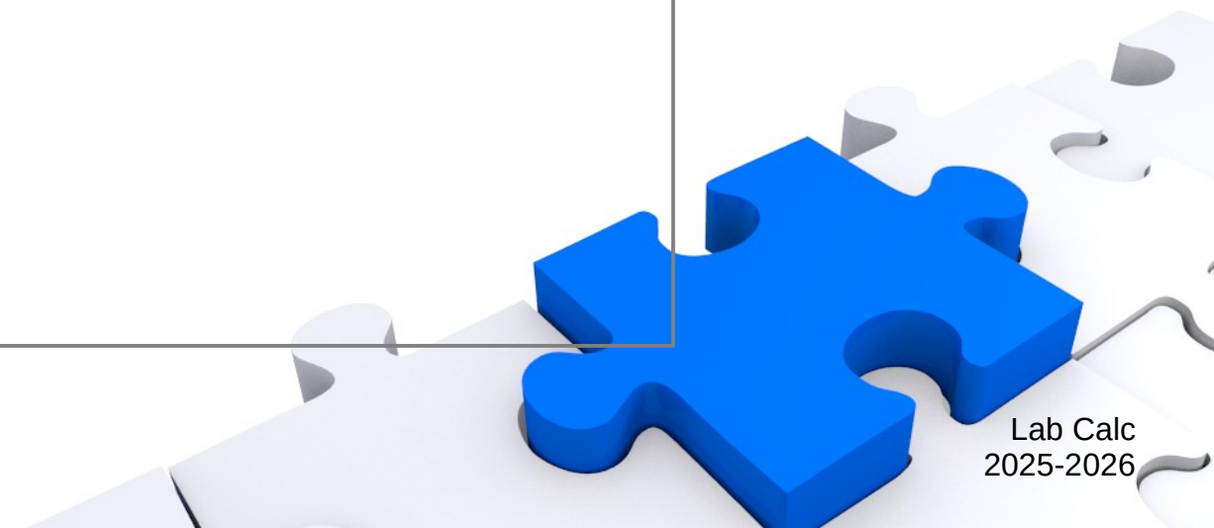
```
In [1]: x = 5; z = 99
In [2]: def addx(y):
   ...:     z = x + y
   ...:     return z
In [3]: addx(10)
Out[3]: 15
In [4]: z
Out[4]: 99
```



The Python Language

- Reusing code: functions
  - Local vs. global variables: scope and lifetime

```
In [1]: x = 5; z = 99
In [2]: def addx(y):
   ...:     global z
   ...:     z = x + y
   ...:     return z
In [3]: addx(10)
Out[3]: 15
In [4]: z
Out[4]: 15
```



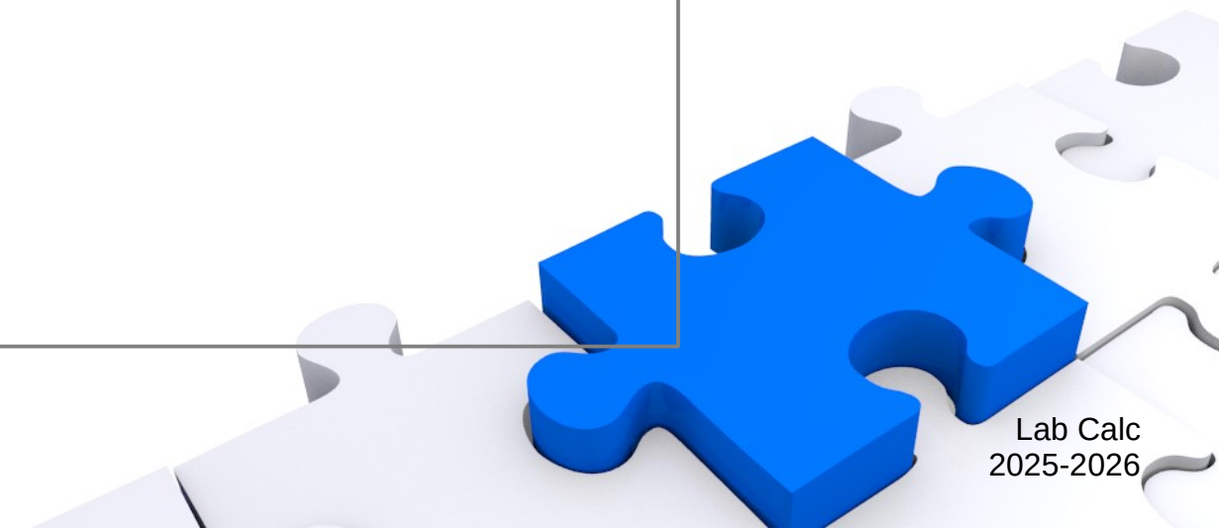
The Python Language

Università di Roma Tor Vergata

- Reusing code: functions
  - Defining anonymous functions

```
funname = lambda arguments: expression
```

```
In [1]: add_one = lambda x: x + 1
In [2]: add_one(2)
Out[2]: 3
In [3]: sum_xy = lambda x, y: x + y
In [4]: sum_xy(2, 3)
Out[4]: 5
```



The Python Language

Università di Roma Tor Vergata

- Reusing code: scripts and modules
  - Python files: filename.py
  - Scripts are top-level files
    - Sequence of python instructions
    - Execute as program in IPython using `%run`
  - Modules are files intended to be (partially) loaded
    - Sequence of functions and variables
    - Load what is needed using `import`

```
import module as name
from module import object as name
```

The Python Language

Università di Roma Tor Vergata

- Reusing code: scripts and modules
  - Example: demo.py

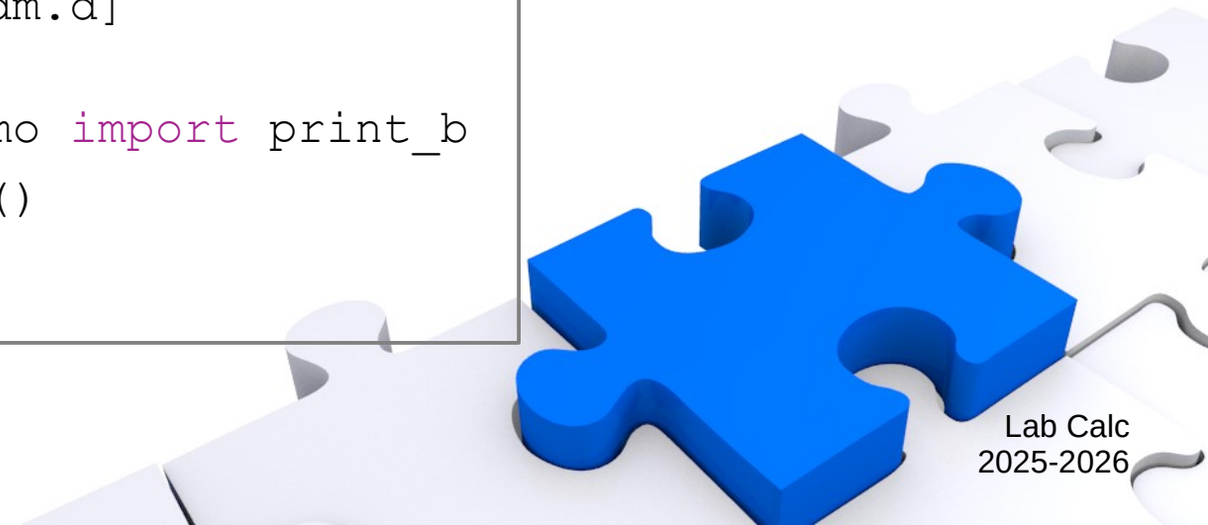
```
#demo file

def print_a():
    print('a')

def print_b():
    print('b')

c = 3
d = 4
```

```
In [1]: import demo
In [2]: demo.print_a()
a
In [3]: import demo as dm
In [4]: [dm.c, dm.d]
[3, 4]
In [5]: from demo import print_b
In [6]: print_b()
b
```



The Python Language

Università di Roma Tor Vergata

- Reusing code: scripts and modules
  - Many built-in modules
    - Mathematics (`math`, `cmath`), system (`os`, `sys`), `...`
  - Mathematical module: `import math`
    - Mathematical constants: `math.pi`, `math.e`
    - Number-theory functions: `math.factorial(x)`, `math.gcd(x, y)`
    - Power functions: `math.exp(x)`, `math.log(x)`, `math.sqrt(x)`
    - Trigonometric functions: `math.cos(x)`, `math.sin(x)`, `math.tan(x)`
  - Package/library: a collection of modules
    - `numpy`, `scipy`, `matplotlib`, `...`