Introduction to Python

1. What is Python?
   - a scripting language
2. Control and Data Structures
   - algorithms and data structures
   - tuple, list, dictionary
   - list comprehensions, sharing references
3. Functions and Modules
   - `lambda` and `def`
   - call by value and call by reference
   - top down functional design
4. Coding Style
   - style guide and `pylint`

MCS 507 Lecture 2
Mathematical, Statistical and Scientific Software
Jan Verschelde, 23 August 2023

Introduction to Python

1. What is Python?
   - a scripting language
2. Control and Data Structures
   - algorithms and data structures
   - tuple, list, dictionary
   - list comprehensions, sharing references
3. Functions and Modules
   - `lambda` and `def`
   - call by value and call by reference
   - top down functional design
4. Coding Style
   - style guide and `pylint`

What is Python?

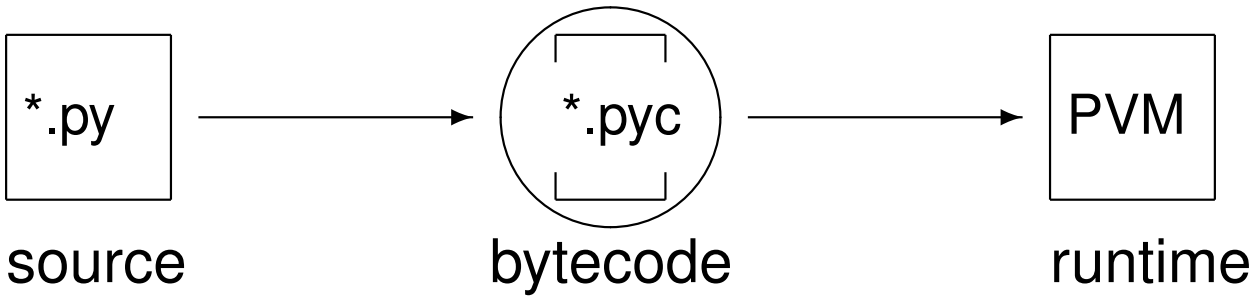
```
>>> print("Hello World!")
Hello World!
```

- dynamically typed, interpreted
- framework language, batteries included
- modular and object oriented

IDLE, REPL, PVM

- Python's Integrated Development Environment (IDE) is IDLE.
- The interpreter runs a Read-Eval-Print Loop (REPL).

The Python Virtual Machine:



The Python interpreter creates bytecode that is then executed by the Python Virtual Machine at runtime.

type and string representation

Every variable has a type:

```
>>> x = 3.1415
>>> type(x)
<class 'float'>
```

and a string representation:

```
>>> str(x)
'3.1415'
```

We can evaluate a string:

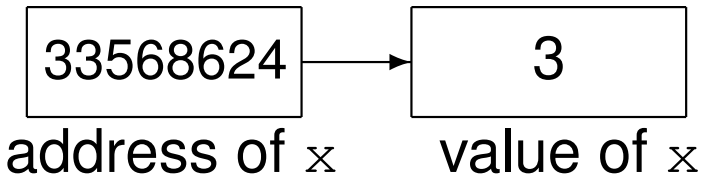
```
>>> sx = str(x)
>>> sx
'3.1415'
>>> eval(sx)
3.1415
```

memory locations and references to objects

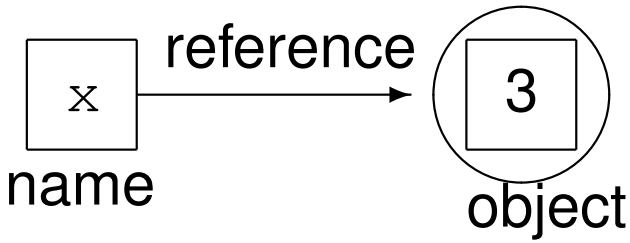
A variable has a value and an address.

```
>>> x = 3
>>> id(x)
33568624
```

The machine view:



In Python, the name `x` *refers to* the object 3.



technical note: dereferencing

In CPython, that is: an implementation of Python based on C, we can *dereference*: obtain the value via its reference.

```
>>> x = 3.1415
>>> ix = id(x)
>>> from ctypes import cast, py_object
>>> cast(ix, py_object).value
3.1415
```

elementary types and composite types

- Every variable has a type.
- The type determines which operations are available.

We distinguish between elementary and composite types.

1. Elementary types: `int`, `float`, `bool`, `str`.
   - Integers are of arbitrary precision.
   - Strings are elementary, there is no character type.
2. Composite types:
   1. tuple: sequence of fixed length, e.g.: `complex`;
   2. list: mutable sequence;
   3. dictionary: lookup or hash table.

Introduction to Python

1. What is Python?
   - a scripting language
2. Control and Data Structures
   - algorithms and data structures
   - tuple, list, dictionary
   - list comprehensions, sharing references
3. Functions and Modules
   - `lambda` and `def`
   - call by value and call by reference
   - top down functional design
4. Coding Style
   - style guide and `pylint`

Algorithms and Data Structures

Niklaus Wirth: programs = algorithms + data structures

Three basic control structures in any algorithm:

1. sequence of statements
2. conditional statement: if else
3. iteration: while and for loop

For every control structure,
we have a matching data structure:

	control structures	data structures
1	sequence	tuple
2	if else	dictionary
3	while / for	list

Introduction to Python

1. What is Python?
   - a scripting language
2. Control and Data Structures
   - algorithms and data structures
   - tuple, list, dictionary
   - list comprehensions, sharing references
3. Functions and Modules
   - `lambda` and `def`
   - call by value and call by reference
   - top down functional design
4. Coding Style
   - style guide and `pylint`

programs transform data

All data are sequences of bits, or bit tuples.

Consider the tuple assignment:

```
>>> (a, b) = (1, 2)
>>> a
1
>>> b
2
```

Swapping the values without temporary variable:

```
>>> (a, b) = (b, a)
>>> a
2
>>> b
1
```

conditions and dictionaries

An `if else` statement for a greeting:

```
>>> import time
>>> hour = time.localtime()[3]
>>> if hour < 12:
...     print('good morning')
... else:
...     print('good afternoon')
...
good morning
```

A dictionary is a set of keys and values. The keys store the possible conditions, while the values hold the arguments of the `print()`.

```
>>> d = { True: 'good morning',
...       False: 'good afternoon' }
>>> d[hour < 12]
'good morning'
```

useful constructions

Python	what it means
`D = { }`	initialization
`D[<key>] = <value>`	add a `key:value` pair
`D[<key>]`	selection of value, given key
`<key> in D`	returns True or False depending on whether D[<key>] exists
`D.items()`	`dict_items` of tuples (key, value)
`D.keys()`	returns `dict_keys` of all keys
`D.values()`	returns `dict_values` of all values

loops and lists

A list of all odd positive numbers less than ten:

```
>>> L = [x for x in range(1, 11, 2)]
>>> L
[1, 3, 5, 7, 9]
```

which is equivalent to

```
>>> L = []
>>> for x in range(1, 11, 2): L.append(x)
...
>>> L
[1, 3, 5, 7, 9]
```

The `range(1, 11, 2)` starts at `1`, ends at `10` (not 11), and proceeds with increments of `2`.

Introduction to Python

1. What is Python?
   - a scripting language
2. Control and Data Structures
   - algorithms and data structures
   - tuple, list, dictionary
   - list comprehensions, sharing references
3. Functions and Modules
   - `lambda` and `def`
   - call by value and call by reference
   - top down functional design
4. Coding Style
   - style guide and `pylint`

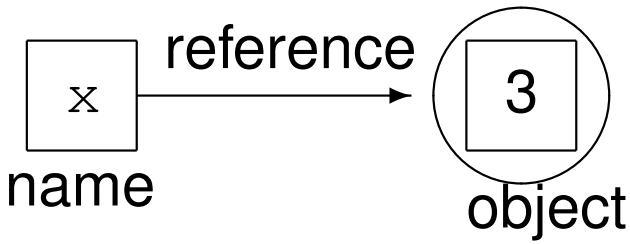
list comprehensions

Suppose we want to list the decimal expansion of π.

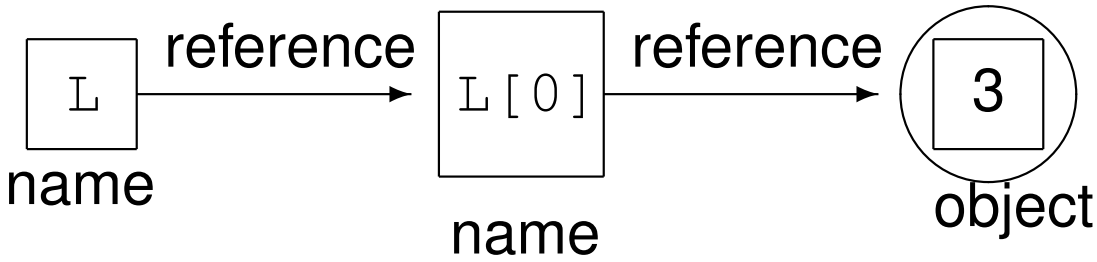
```
>>> from math import pi
>>> pi
3.141592653589793
>>> spi = str(pi)
>>> type(spi)
<class 'str'>
>>> slice = spi[2:10]
>>> slice
'14159265'
>>> d = [c for c in slice]
>>> d
['1', '4', '1', '5', '9', '2', '6', '5']
>>> [int(x) for x in d]
[1, 4, 1, 5, 9, 2, 6, 5]
```

lists and references

```
>>> x = 3
```

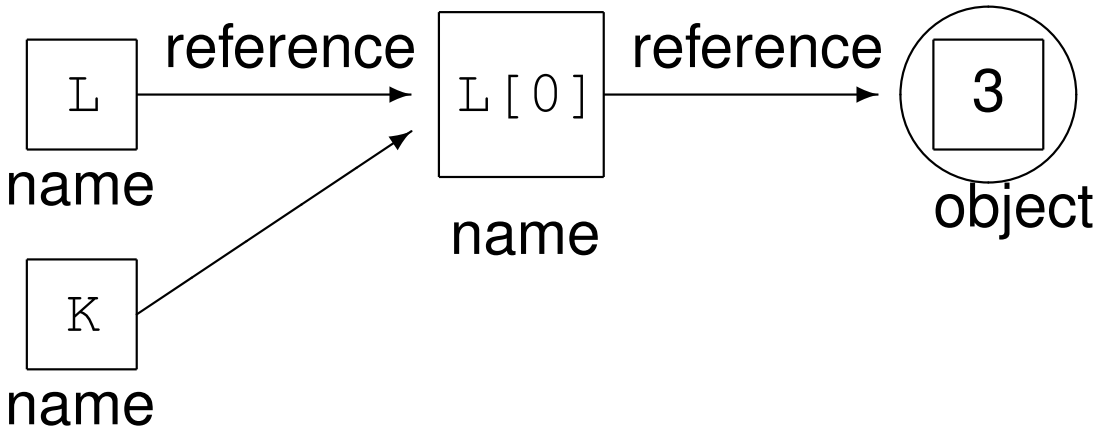



```
>>> L = [3]
```



sharing references

```
>>> L = [3]
>>> K = L
```




```
>>> K[0] = 4
>>> L
[4]
```

We changed `L` through `K`.

Introduction to Python

1. What is Python?
   - a scripting language
2. Control and Data Structures
   - algorithms and data structures
   - tuple, list, dictionary
   - list comprehensions, sharing references
3. Functions and Modules
   - `lambda` and `def`
   - call by value and call by reference
   - top down functional design
4. Coding Style
   - style guide and `pylint`

defining functions

One line definitions with `lambda`:

```
fun = lambda x: x**2
```

With `def`, the above is equivalent to

```
def fun(x):
    """
    Returns the square of x.
    """
    return x**2
```

Every function has

1. a name, input and output arguments may be empty
2. a documentation string (optional, recommended)
3. a definition, with or without return

Introduction to Python

1. What is Python?
   - a scripting language
2. Control and Data Structures
   - algorithms and data structures
   - tuple, list, dictionary
   - list comprehensions, sharing references
3. Functions and Modules
   - `lambda` and `def`
   - call by value and call by reference
   - top down functional design
4. Coding Style
   - style guide and `pylint`

call by value and call by reference

call by value: a function takes the value on input

```
>>> f = lambda x: x**2
>>> f(3)
9
```

and does not change the input variable.

call by reference: a reference is passed and the input may change

```
>>> def fun(arg):
...     arg[0] = arg[0]**2
...
>>> L = [3]
>>> fun(L)
>>> L
[9]
```

an illustration of call by reference

The class `list` exports the method `sort`.

In the object-oriented terminology, the application of

- the method `sort`
- to a list `L`

is written as `L.sort()`.

We cannot write `sort(L)`, but we may define a function:

```
>>> mysort = lambda L: L.sort()
>>> L = [3, 1, 4, 1, 5]
>>> mysort(L)
>>> L
[1, 1, 3, 4, 5]
```

The `mysort` takes on input a list, a reference to the sequence of numbers stored in the list and sorts the list.

Introduction to Python

1. What is Python?
   - a scripting language
2. Control and Data Structures
   - algorithms and data structures
   - tuple, list, dictionary
   - list comprehensions, sharing references
3. Functions and Modules
   - `lambda` and `def`
   - call by value and call by reference
   - top down functional design
4. Coding Style
   - style guide and `pylint`

modules

A module is a collection of functions.

In top down functional design, we start with the main function:

1. `input = prompt_user()`
2. `output = compute(input)`
3. `write_report(output)`

The functions `prompt_user()`, `compute()`, and `write_report()` are defined in the same file that holds the function `main()`.

When using the Jupyter notebook, there is no more prompting the user.

The `path` variable in the `sys` module lists the directories the interpreter searches for the modules.

Introduction to Python

1. What is Python?
   - a scripting language
2. Control and Data Structures
   - algorithms and data structures
   - tuple, list, dictionary
   - list comprehensions, sharing references
3. Functions and Modules
   - `lambda` and `def`
   - call by value and call by reference
   - top down functional design
4. Coding Style
   - style guide and `pylint`

Python coding style

PEP 8 – Style Guide for Python
PEP = Python Enhancement Proposal
available at `http://www.python.org/dev/peps/pep-0008/`

This document gives coding conventions for Python code.
The guidelines are intended to improve readability of the code.

For example in *Names to Avoid*:

Never use the characters 'l' (lowercase letter el), 'O' (uppercase letter oh), or 'I' (uppercase letter eye) as single character variable names.

In some fonts, these characters are indistinguishable from the numerals one and zero. When tempted to use 'l', use 'L' instead.

checking Python code

`pylint` is a static checker of Python code.

Copied from `www.pylint.org`:

Pylint is a Python source code analyzer which looks for programming errors, helps enforcing a coding standard and sniffs for some code smells (as defined in Martin Fowler's Refactoring book).

Pylint has many rules enabled by default, way too much to silence them all on a minimally sized program. It's highly configurable and handle pragmas to control it from within your code. Additionally, it is possible to write plugins to add your own checks.

It's a free software distributed under the GNU Public Licence.

Summary + Exercises

We took a tour of the Python scripting language.

Exercises: (1 is exercise 1.12 *how to cook the perfect egg*)

1. The time t in seconds it takes for the center of the yolk to reach the temperature T_y in Celsius is given by the formula

$$t = \frac{M^{2/3} c \rho^{1/3}}{K \pi^2 (4\pi/3)^{2/3}} \ln \left[0.76 \frac{T_o - T_w}{T_y - T_w} \right]$$

where M is the mass, ρ the density, c is the specific heat capacity, and K is the thermal conductivity of the egg. Typical values for a large egg are $M = 67$ g, $\rho = 1.038$ g cm^{-3}, $c = 3.7$ J g^{-1} K^{-1}, and $K = 5.4 \cdot 10^{-3}$ W cm^{-1} K^{-1}. The original temperature in Celsius is T_o and $T_w = 100$ C is the temperature of the boiling water. Write a script `egg.py` using the formula to compute the time t it takes for $T_y = 70$ C. Prompt the user for the value of T_o (in script), or set: $T_o = 4$ C (fridge), $T_o = 20$ C (room temperature).

more exercises

2. Do `from math import cos, pi`
   and consider `cos(1)`, `cos(1+2*pi)`, `cos(1+4*pi)`,
   `cos(1+k*pi)`, for increasing even values for `k`,
   take `k` at least equal to 100, or larger.
   Compute the increase of roundoff errors.
   Can you derive a relation between `k` and the error?
   - Consider the Taylor expansion of $\cos(1 + k\pi)$.
   - Is there a difference with the `numpy.cos()`?

3. Consider the quadratic formula $x = \dfrac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ for the roots of $ax^2 + bx + c$. Write an interactive Python script that prompts the user for a, b, c and prints the roots.
   For $\sqrt{\cdot}$, use the `sqrt` of `cmath` as that `sqrt` function returns complex numbers (unlike the `math.sqrt`).