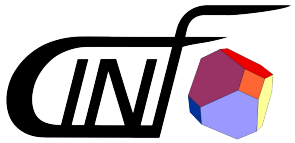


Center for Individual Nanoparticle Functionality

- Introduction
- What is a computer program
- Math
- Types
- Variables
- Containers
- Loops
- Boolean expressions
- Control flow (if-elif-else)
- Functions
- Summary

Python part 0

Center for Individual Nanoparticle Functionality

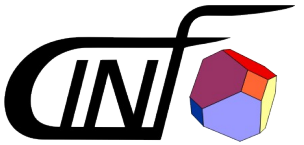
- Introduction

Python part 0

The program

- Move through each of the topics in the outline
- For each topic there will be:
  - A presentation by me
  - A “type along” session, where we type the new things together
  - A few exerciser (the exercises are meant mainly for typing repetition, so most are simple)
- There will also be a few short detours along the way

Why program? To do cool stuf



- To do data treatment (reproducably)
- Talk to equipment - to use it for things that the manufaturer did not think of
- To combine data from different sources
- To produce good looking graphs
- To automate tedious and/or time consuming tasks
- And because its fun – programming is like Lego for adults

Why program? For reproduceability

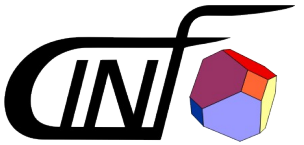
- As scientists we often rely need to:
  - Repeat the same data treatment for different data sets
  - Repeat a procedure
  - Repeat repeat repeat
- But people are not very good at repeating simple tasks (they get bored or forget)
- Computers on the hand are **great** at repeating
- Why not let them do it

The “super coder” myth

- That programming skills are somehow special and something that only people with a predeposition can do
- That skills are distributed in a bimodal fashion with only super-ninjas and those that cannot not at all

FALSE!

The “super coder” myth is bogus



- Programming is a skill just like any other and anyone can develop along the skill levels
- It takes teaching, repetition and practice, just like any other skill
- Relax, enjoy learning something new and don't panic, if you do not feel able to program facebook from scratch by the end of the day
- **That's normal!**

A bit about wording

- I have chosen to be accurate with wording
- Might seem a little abstract
- E.g:
  - **Object** An object “is a thing” in a program
  - **Sequence** A sequence is “a string of somethings”
  - etc.

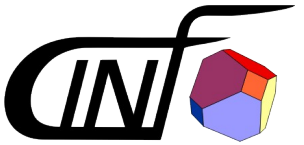
Python

- Will use Python
  - Ease of use
  - Emphasis on readability
  - Popularity in the scientific community
- MANY or all of the concepts we talk about are general and used in many programming environments

- What is a computer program
- Math
- Types
- Variables
- Containers
- Loops
- Boolean expressions
- Control flow (if-elif-else)
- Functions

Python part 0

What is a computer program



- A set of instructions
- Executed in order
- Needs to be correct, follow a syntax
- Think of it like ...
- A cooking recipy
- Give a chef instructions one at a time

Directions

- 1. In a large bowl, sift together the flour, baking powder, salt and sugar. Make a well in the center and pour in the milk, egg and melted butter; mix until smooth.
- 2. Heat a lightly oiled griddle or frying pan over medium high heat. Pour or scoop the batter onto the griddle, using approximately 1/4 cup for each pancake. Brown on both sides and serve hot.

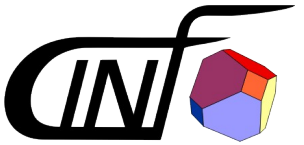
http://allrecipes.com/recipe/good-old-fashioned-pancakes/

What is a computer program

```
with sift:
    apply flour
    apply baking powder
    apply salt
    apply sugar
make center well
apply egg
apply milk
apply melted butter
while not well mixed:
    mix with spoon
```

Pseudo-code to make pancake batter. Yum!

The interpreter



- Usually programs are written as sequences of instructions in a file
- In Python (unlike all languages) you actually can give it instructions *live*
- For this you use the "*the interpreter*"
- Usually used to experiment or test
- We will use it to learn interactively

Center for Individual Nanoparticle Functionality

- What is a computer program
- Math
- Types
- Variables
- Containers
- Loops
- Boolean expressions
- Control flow (if-elif-else)
- Functions

Python part 0

Math

- All the usual suspects
- Except division, may not do what you want
- Order of operations as you would expect

  PEMDAS

Operator	Action
+	Plus
-	Minus
/	Division (careful)
*	Multiplication
**	Raise to power
( )	Used for precedense

Math

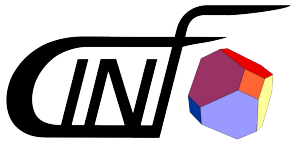
- With division / between integers performs flooring integer division
  - e.g. 3/4=0
- To turn it into float, convert one to float first
  - float(3)/4 = 0.75
- Can use exponential notation

  4E-8 = 4 * 10^-8
- Let's get started typing

Start the Python interpreter from your program menu

Math

Type along session in the terminal:
Math



Center for Individual Nanoparticle Functionality

- What is a computer program
- Math
- Types
- Variables
- Containers
- Loops
- Boolean expressions
- Control flow (if-elif-else)
- Functions

Python part 0

Types

- In computer programs, items (called objects) have **a type**
- The type determines:
  - What data it can contain
  - How it interacts with other objects of the same or a different type
    - In short, what you can do with it
  - (If it can be changed)
- In short, an objects type defines its functionality

Type: Shoe

Type: Box

Type: Integer

http://www.clipartpanda.com/clipart_images/shoes-clip-art-5293230
http://www.clipartpanda.com/categories/wooden-box-clipart
http://4vector.com/free-vector/number-in-green-rounded-square-clip-art-115333

Types

- You have already met a few types
- The most fundamental are:

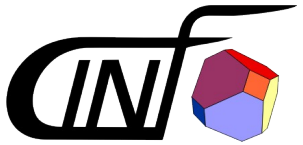
float is short for floating point number and is the universal representation for fractional numbers in science.

Also sometimes referred to as
single → single precision float
double → double precision float

Type	Example
int	47
float	47.0
string	“fourtyseven”
bool	True
NoneType	None

This one we haven't seen before. It means:
“nothing to see here”

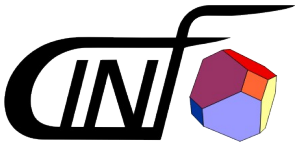
The *type* function



- The *type* function can be used to get the type of an object

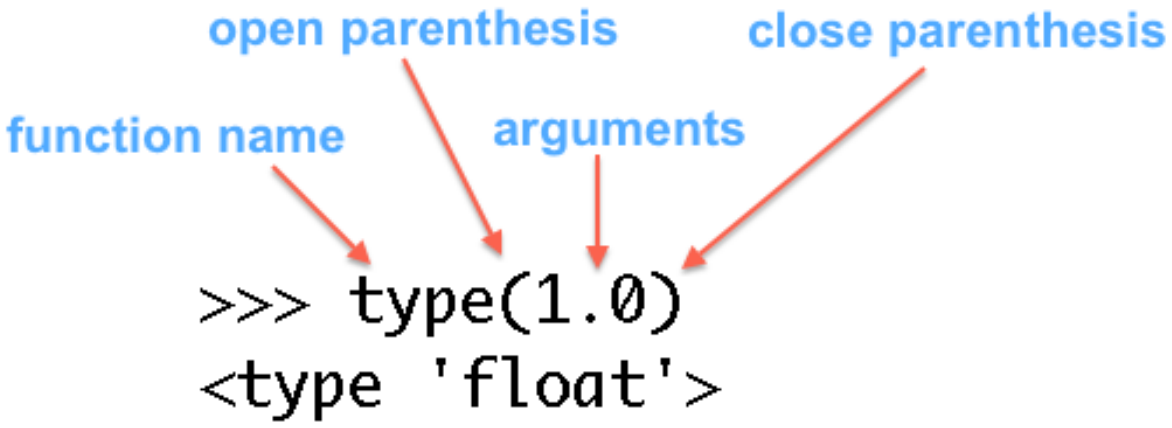
```
>>> type(47)
<type 'int'>
>>> type(47.0)
<type 'float'>
>>> type('fourtyseven')
<type 'str'>
>>> type(True)
<type 'bool'>
```

Wait, what is a function?



- A function gathers useful and re-useable bits of work together
- A function:
  - Takes input and produces output (just like in math)
  - And/or can perform tasks
- The different inputs to a function are called *“arguments”*
- Functions can be *“called”* or *“invoked”*
- Function can *“be given”* or *“passed”* arguments

The anatomy of a function call



`type` is a function that "takes" one argument
we are "calling" `type` with the argument 1.0
we can "pass" any Python object to `type`

https://openhatch.org/wiki/Boston_Python_Workshop_8/Friday/Tutorial#Types

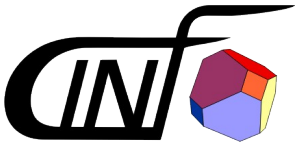
Types – Type along

Type along session in the terminal:
Types

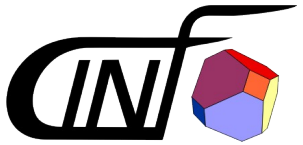
Types - Exercises

- Get the type of different objects with **type()**
- Try math on objects that you would not expect to work
  - E.g. multiply a strings, booleans and numbers
  - Does it work?
  - What is the type
  - Is there a logic to it?

The command history



- Try and press the up arrow in the interpreter
- The interpreter keeps a history of what you have entered
- Makes it easy to go back and repeat actions
- Simple press *up arrow* untill the correct command and then *enter*



Working with strings

Working with strings

- Strings are fundamental (also for scientists)

- 'This is a string'
- To concatenate strings use +
- To add the string representation of an object to a string, use str(object)
- ' and “ are synonomous
- The used quote needs to be escaped inside the strings

Working with strings

```
>>> 'The start ' + 'and the end'
'The start and the end'

>>> 'Concatenating a number to the string ' +
str(1)
'Concatenating a number to the string 1'

>>> 'It\'s necessary to quote'
"It's necessary to quote"

>>> "It's necessary to quote"
"It's necessary to quote"
```

Strings – Type along

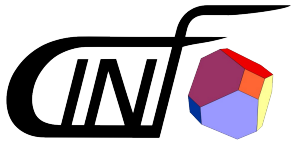
Type along session in the terminal:
Types

Strings - Exercises

- Exercise 1-4 on

  http://www.codecademy.com/courses/python-beginner-en-kSQwt/0/1

  (Keep this page open for later)

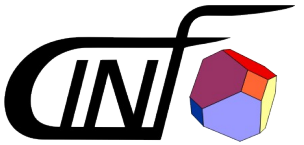


Center for Individual Nanoparticle Functionality

- What is a computer program
- Math
- Types
- Variables
- Containers
- Loops
- Boolean expressions
- Control flow (if-elif-else)
- Functions

Python part 0

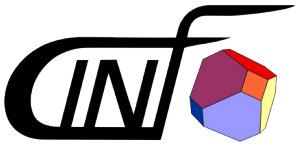
Variables



- A variable **is a nametag** for an object
- Almost like math variables
- Can “be” of **any type**
- Giving an object a name is called *assignment*

```
>>> x = 47
>>> x
47
>>> type(x)
<type 'int'>
>>> 2 * x
94
```

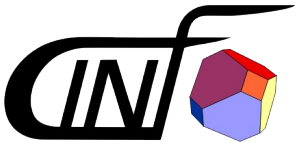
Mix and match, variables and object



- You can mix variables with objects in expression
- Think if it as if variables are replaced by the object they refer to

```
>>> a = 22
>>> b = 2
>>> the_answer = 2 * a - b
>>> the_answer
42
```

Assignment to a variable



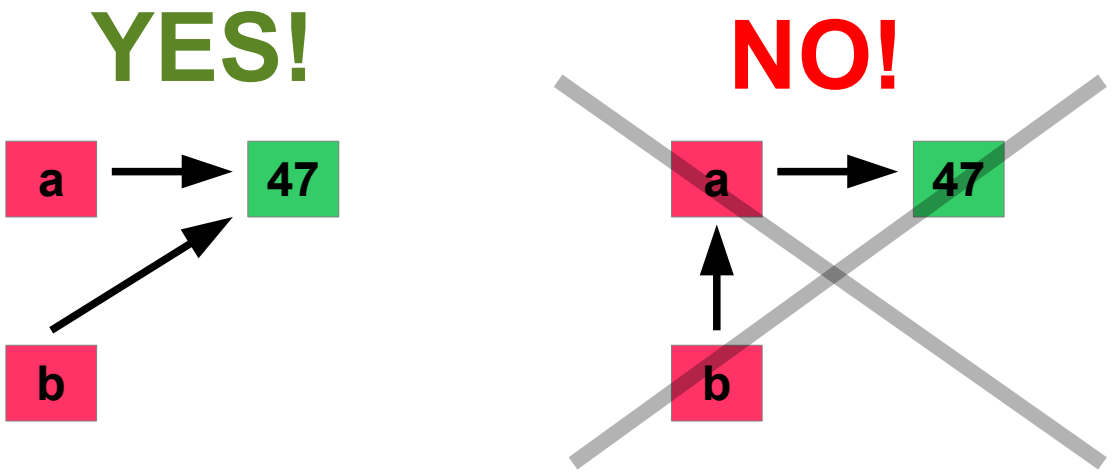
- What happens if you try and assign a variable to a varaible?

```
a = 47
b = a
```

Assigment *to* a variable

- You ***cannot*** assign a variable to another variable
- It will fall through and point to the same object

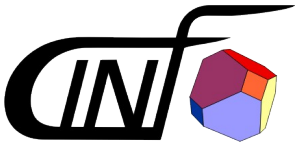
```
>>> a = 47
>>> b = a
```



Variables

- Variable names can only contain letters and underscores (no spaces)
- Keep consistent, use lowercase underscore
  - measurements
  - measurement_set
  - number_of_measurements
- Must be descriptive, to help yourself read your program
  - Yes: number_of_points, length_of_queue etc.
  - No: number, length, size, my_integer etc.

About output



- About of an expression will be printed to screen …
- … unless assigned to a variable
- Functions can **both change the value of something that you pass in and return it**

```
>>> 4
4
>>> a = 4
>>> a
4
```

Variables – Type along

Type along session in the terminal:
Variables

Variables - Exercises

- Assign objects of different types to variables
- Try expressions with variables only
- Try expressions with a combination of variables and objects
- Assign a number to a variable (a=47),
  - assign a new vairable to that first variable (b=a)
  - check the value of a and b
  - change the value of a
  - check the value of a and b

Center for Individual Nanoparticle Functionality

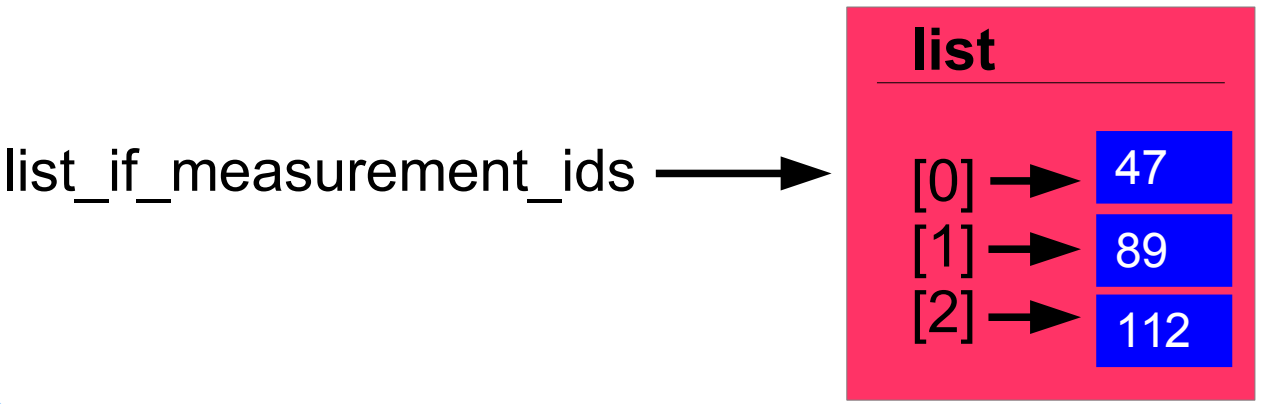
- What is a computer program
- Math
- Types
- Variables
- Containers
- Loops
- Boolean expressions
- Control flow (if-elif-else)
- Functions

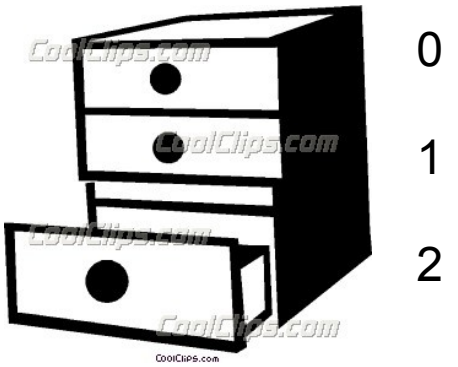
Python part 0

Containers

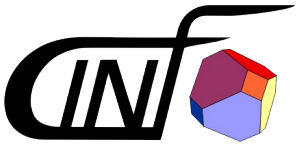
- Creating a variable for every measure that you need is impractical
- Containers can **contain** several objects and make it easy to access them
- Containers are in themselves objects

  that a variable can point to



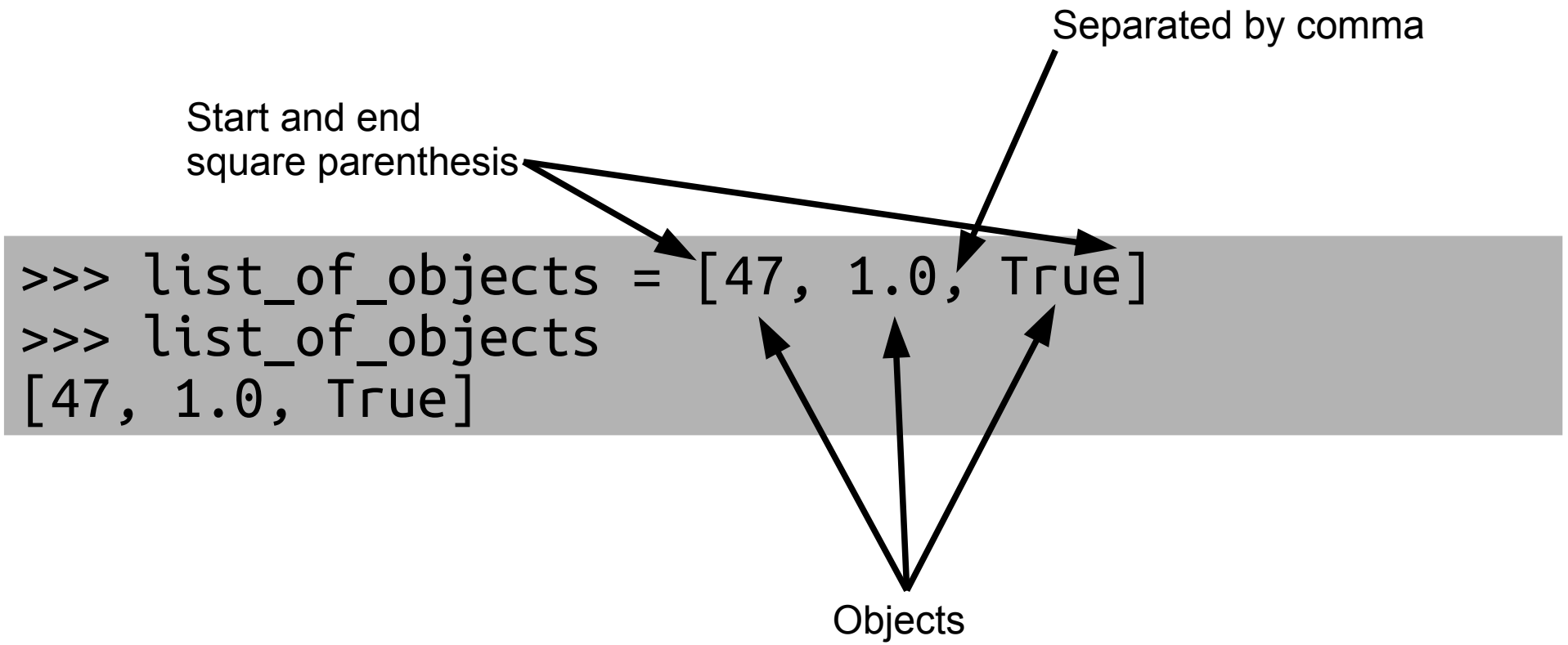


The simples container – the **list**

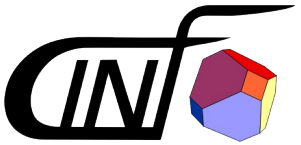


- **Ordered:** Objects are in order in *slots*
- **In-homogenuous:** A list can contain a mix of different types of objects
- Objects are assigned and retrieved with integer indeces
- Create a list by typing [] or calling list() on a sequence

A list


Separated by comma
Start and end
square parenthesis
>>> list_of_objects = [47, 1.0, True]
>>> list_of_objects
[47, 1.0, True]
Objects

Indexing lists



- To index a list use the [*index_number*]
- Index numbers are 0-based (first one is 0)

```
>>> list_of_objects = [47, 1.0, True]
>>> list_of_objects[0]
47
>>> list_of_objects[2]
True
>>> list_of_objects[0] = 7
>>> list_of_objects
[7, 1.0, True]
```

List – Type along

Type along session in the terminal:
List

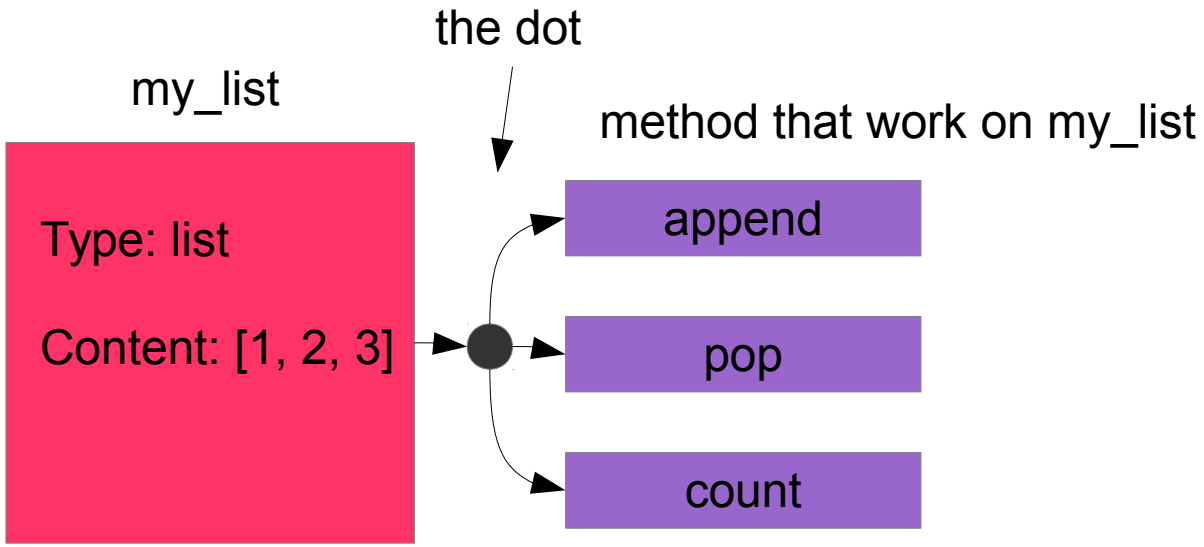
Add elements in a list

- Primarily with *append*

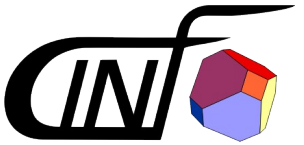
```
>>> list_of_objects
[7, 1.0, True]

>>> list_of_objects.append(9)
>>> list_of_objects
[7, 1.0, True, 9]
```

Hold on, what is a method?

- A method is a functions that:
  - Is attached to an object
  - Performs actions on this object
- All objects have methods, even ints



A list has quite a few methods



- Call the *dir* function of an object to find its methods
- **append** adds an object to the end
- **count** counts occurences of an object
- **index** returns first index of an object
- **insert** insert object at index
- **remove** removes first accurence of object
- **pop** remove and return from index
- … more

List – Type along

Type along session in the terminal:
List

List - Exercises

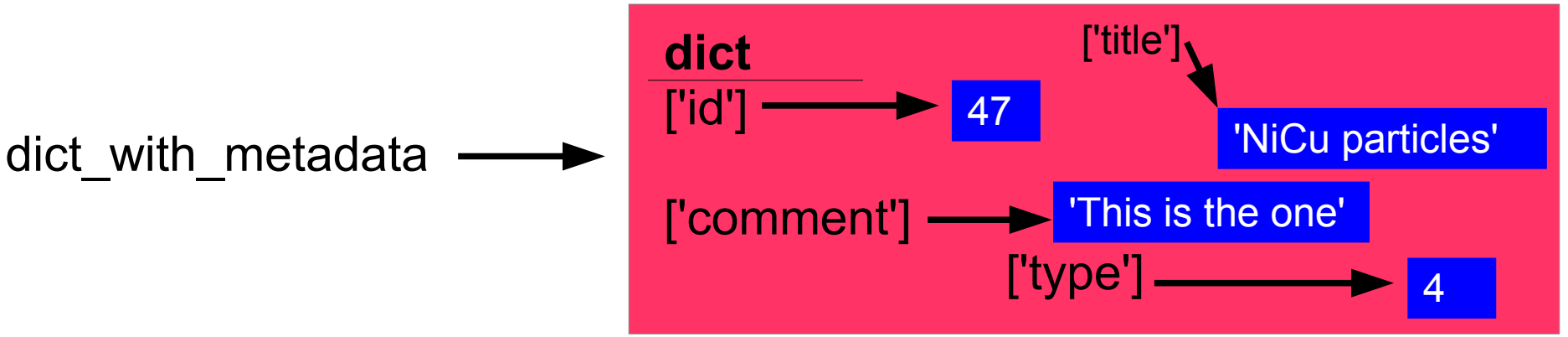
- Make a list with [] and with list() (e.g. by calling it in a string)
- Get objects out and save new values by index
- Try and write two integers as index separated by ':' e.g. mylist[1: 3]
- Make a list with two elements in and try:

  my_variable_a, my_variable_b = mylist
- Try the index and count methods
- Exercises 9 and 11 on codeacademy

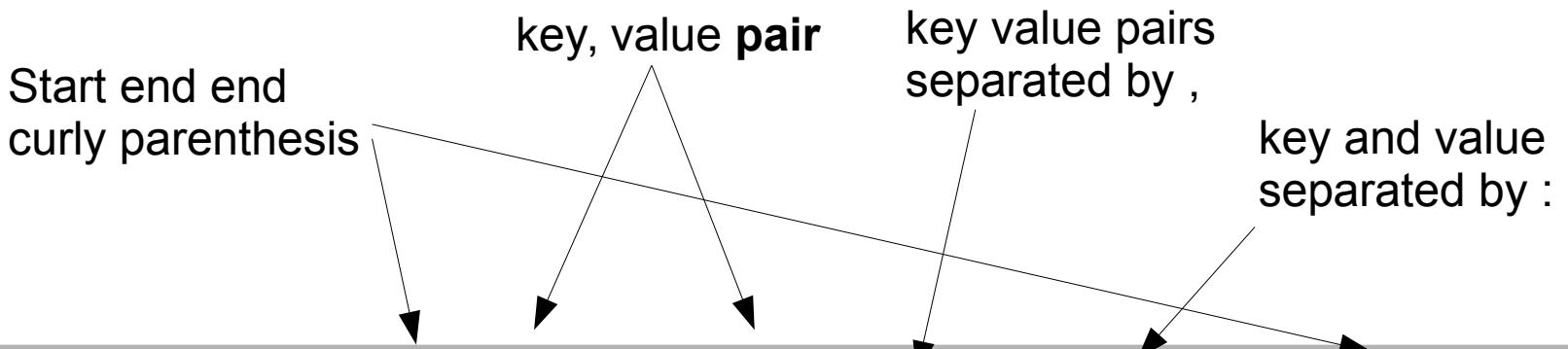
- Call dir and help e.g. on a list and a string and read a litte about some of the methods

A mapping container – the **dict**

- Short for *dictionary*
- An **un-ordered mapping** of keys to values
- Objects are retrieved or assigned with keys
- Create a dict with {} or by calling dict() on a sequence of two-object sequences



Creating a dict




```
>>> mydict = {key0: value0, key1: value1}
>>>
>>> dict([[key0, value0], [key0, value0]])
```

Index by keys

- Index a dict by **a key**
- Like look up word in a dictionary
- A key can be any object that cannot be changed

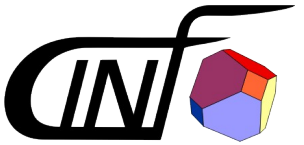
```
>>> mydict = {'title': 'CuZn', 0: True}
>>> mydict
{0: True, 'title': 'CuZn'}
>>> mydict[0]
True
>>> mydict['title']
'CuZn'
```

Also assign by key

- To assign a new value to an existing key, use: **[key]**

```
>>> mydict
{0: True, 'title': 'CuZn'}
>>>
>>> mydict['title'] = 'CuZn nano particles'
>>>
>>> mydict
{0: True, 'title': 'CuZn nano particles'}
>>> mydict['title']
'CuZn nano particles'
```

Dicts has lots of useful methods



- **keys()** list of all the keys
- **values()** list of all the values
- **items()** list of all the key-value pairs
- **pop()** return value and remove it
- **update()** update values with values from another dict
- **get()** is like [] but does not give an error if the key is not in the dict
- … and many more ...

Dict – Type along

Type along session in the terminal:
Dict

Dict - Exercises

- Make a dict by entering it with {}
  - Try and use different types as keys and values
- Get objects out by indexing
- Assign a new value to one of the keys
- Try the *keys*, *values* and *items* methods
- Make a new dict that share some keys and has a few new ones compared to the first one and try and use the *update*
- Try *len()* on a dict
- Call *dir* and *help* on a dict and read a litte about some of the methods

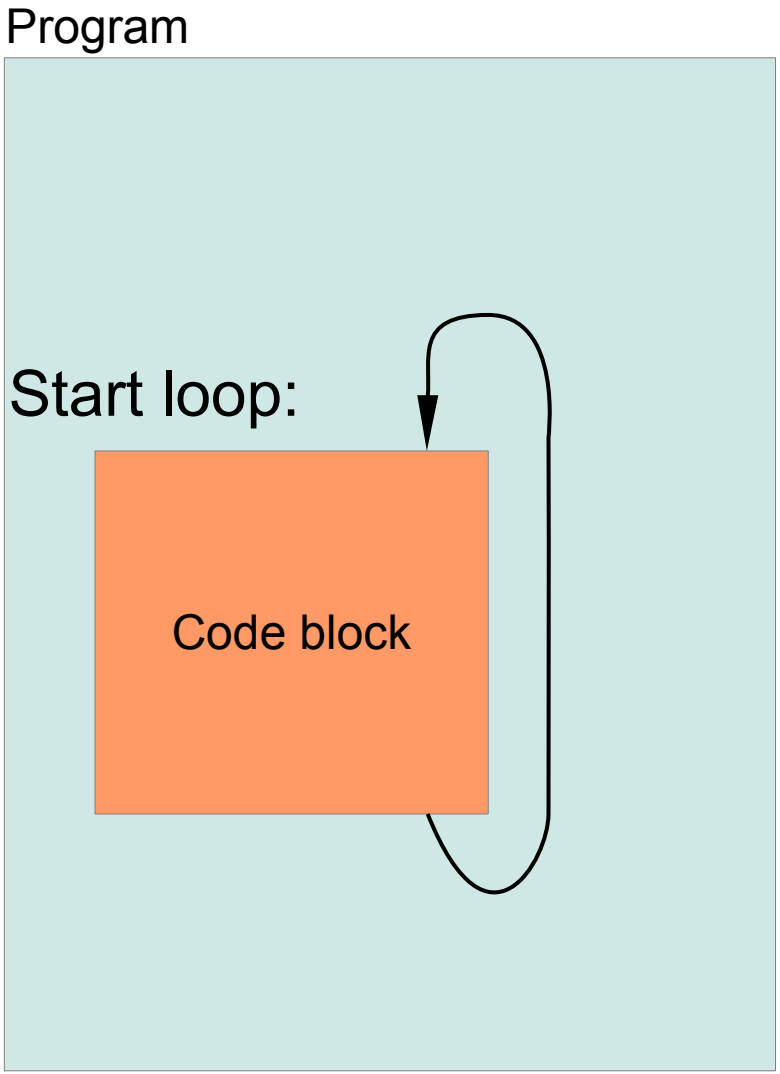
Center for Individual Nanoparticle Functionality

- What is a computer program
- Math
- Types
- Variables
- Containers
- Loops
- Boolean expressions
- Control flow (if-elif-else)
- Functions

Python part 0

Loops

- Loops are fundamental to programming
- Repeat a part of program with different input
- E.g. once for every data set
- Python makes it easier



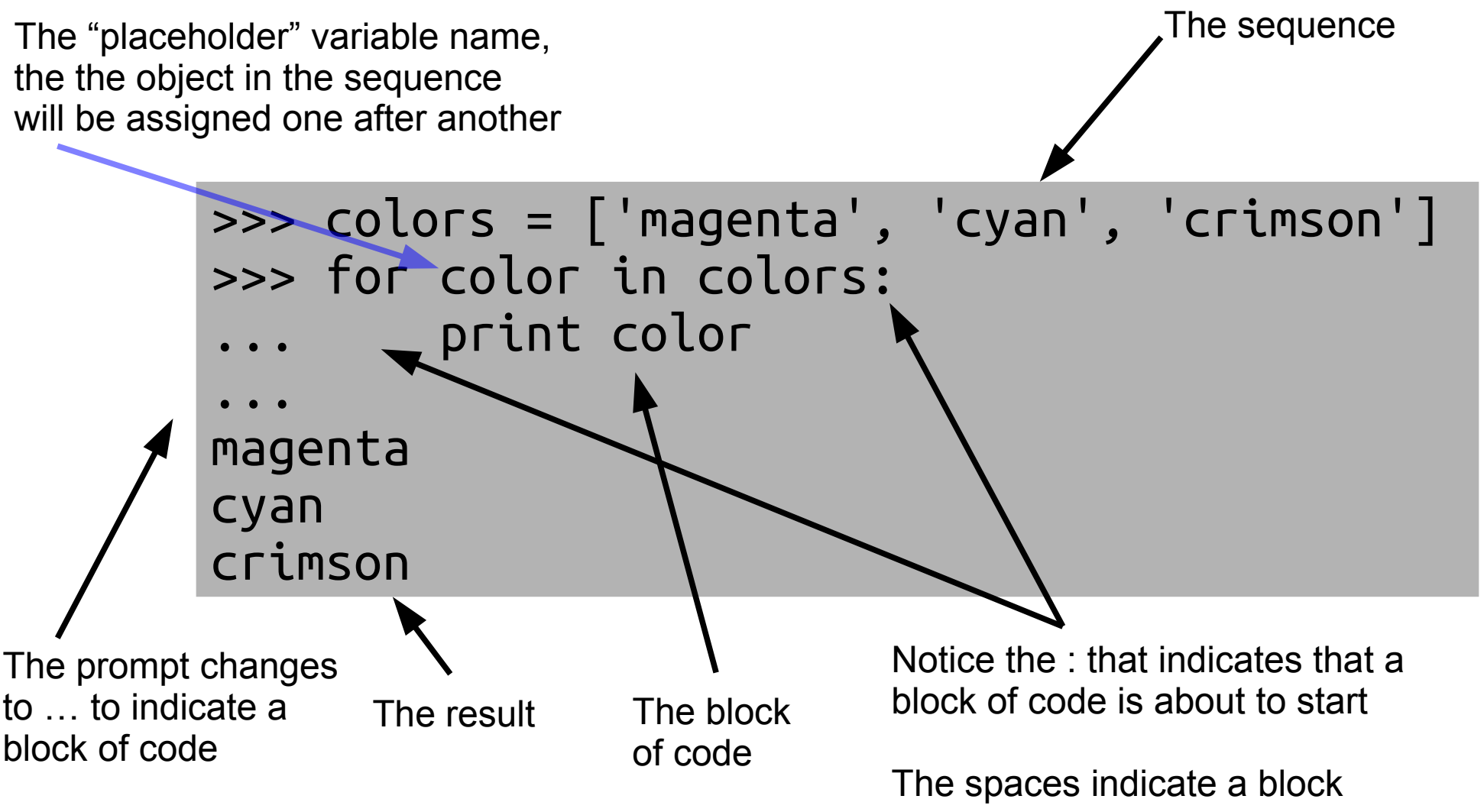
Loops – **for**

- The **for** loop
- Execute the block once for every object in sequence
- And make that object available in code block
- Used whenever you want to do the same thing for a sequence of somethings
- Think: *foreach measurement in my set, analyse and plot*

Loops **for** - example

```
>>> colors = ['magenta', 'cyan', 'crimson']
>>> for color in colors:
...     print color
...
magenta
cyan
crimson
```

Loops **for** - example



About **code blocks**

- Indicates a “group on lines of code”
- That belongs to the line just before
- Sometimes done with parenthesis – in Python we use indentation

The : indicates the beginning of a block

```
for color in colors:
....mystr = color + “ yeah”
....print mystr
```

That block will continue as long as the lines are indented

Use 4 spaces for indentation. Your editor will help you, just press tab

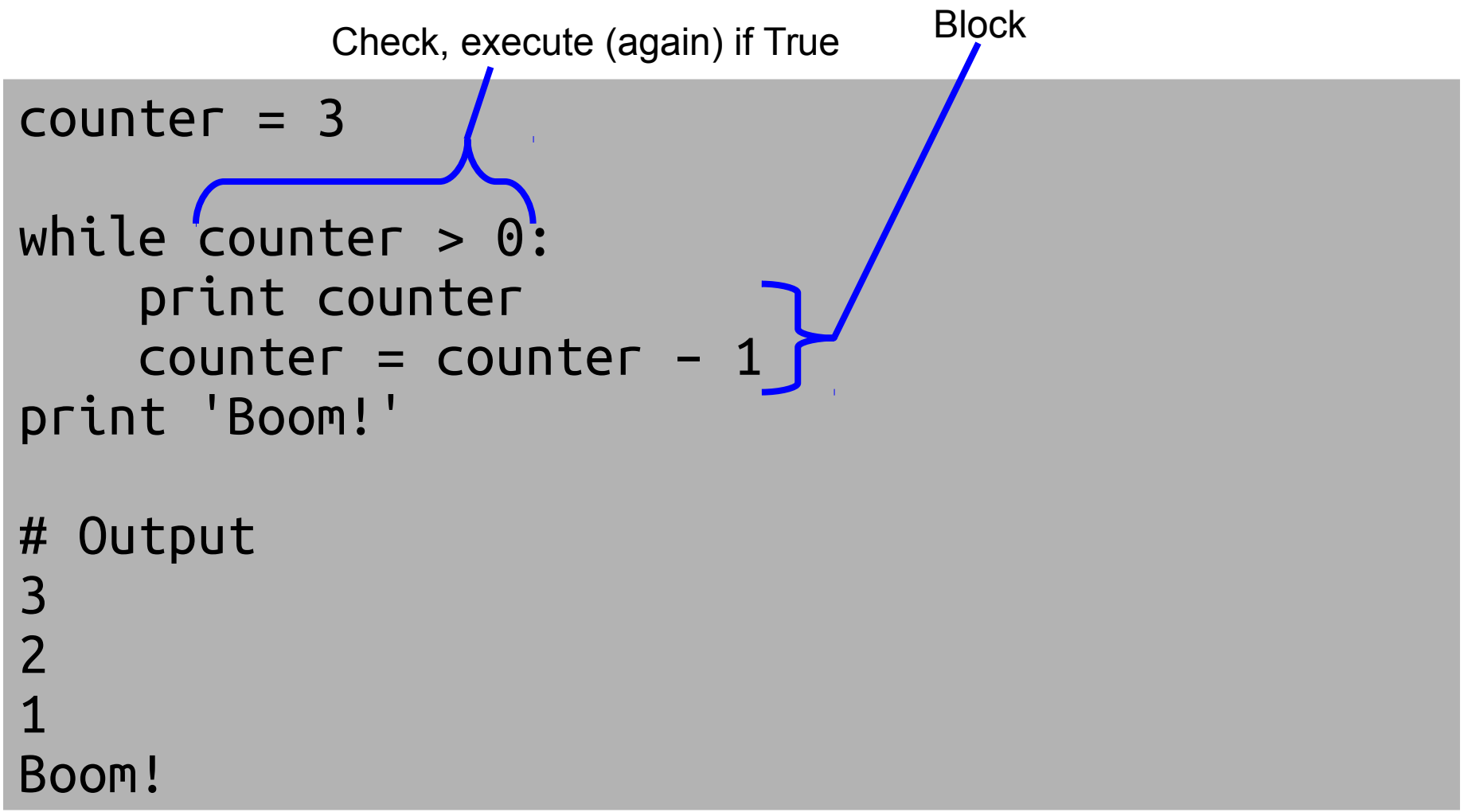
Loops – Type along

Type along session in the terminal:
Loops

Loops - while

- The **while** loop
- Execute the block for as long as an expression evaluates as True
- Used e.g. when one needs to check if something is complete
- Think "*keep doing this as long as ...*"

Loops – while example

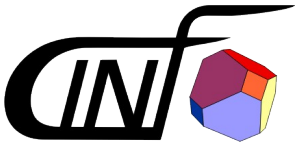


Loops – while, more examples

- Empty a container

```
>>> colors = ['magenta', 'cyan', 'crimson']
>>> while len(colors) > 0:
...     print colors.pop()
...
crimson
cyan
magenta

>>> colors
[]
```

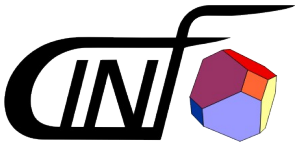
Loops – while, more examples



- Loop forever

```
>>> while True:
...     # Keep doing this until the end of time
...     print “I am still here”
...
I am still here
I am still here
I am still here
I am still here
I am still here
```

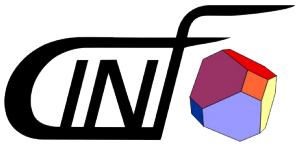
About comments



- Comments are lines of code that are not executed
- Used to explain the program to yourself
- A comment is made by starting a line with #

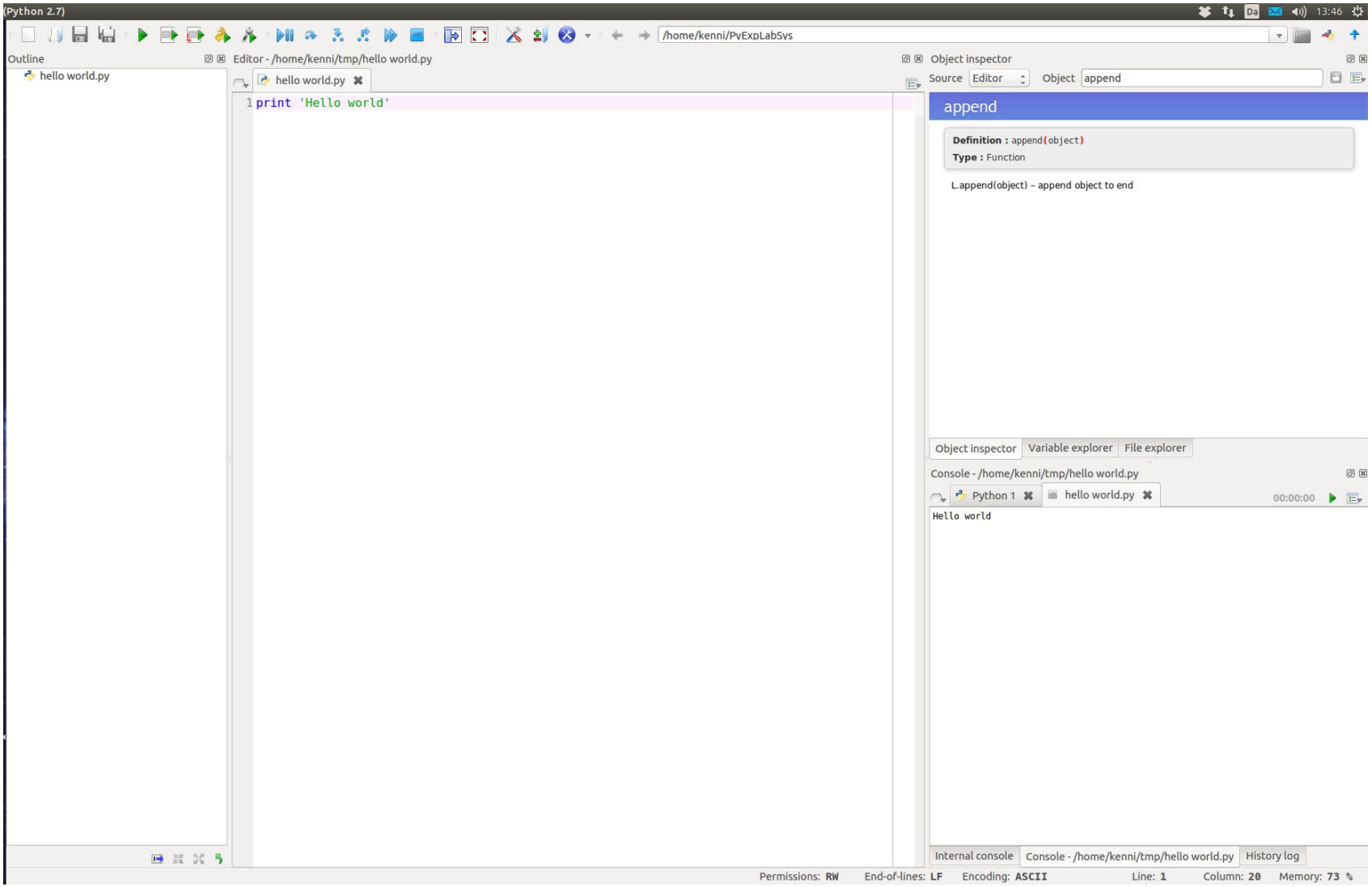
```
# Perform super fancy calculation
result = 2 + 2
```

Programming Python in a file



- The interpreter is really good for trying things out, but we do not want to keep typing the same things over and over again
- Python code can also be executed from a file
- We will write these files with an text editor called “Spyder”

Spyder


(Python 2.7)
13:46
/home/kenni/PvExpLabSys
Outline
hello world.py
Editor - /home/kenni/tmp/hello world.py
hello world.py
1 print 'Hello world'
Object inspector
Source Editor
Object append
append
Definition : append(object)
Type : Function
L.append(object) - append object to end
Object inspector
Variable explorer
File explorer
Console - /home/kenni/tmp/hello world.py
Python 1
hello world.py
00:00:00
Hello world
Internal console
Console - /home/kenni/tmp/hello world.py
History log
Permissions: RW
End-of-lines: LF
Encoding: ASCII
Line: 1
Column: 20
Memory: 73 %

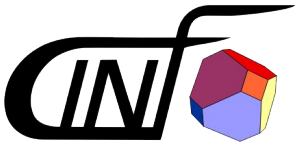
Exercises now in spyder

All type along sessions and exercises from now on are in Spyder

Loops – Type along

Type along session in the terminal:
Loops

Loops – Exercises **IMPORTANT!!!**



- Make a list of strings and make a new list, by looping over the old, modify the strings and append it to the new list
- Make a variable with the value 1. Make a while loop that keeps multiplying it with 0.5 until the result is below 1E-8. Print out the result: 7.450580596923828e-09
- Work through problems 12-19 on codeacademy

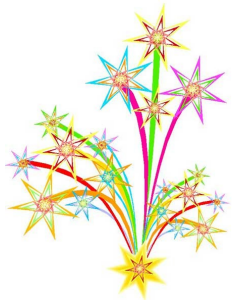
“Loop like a native”

https://www.youtube.com/watch?v=EnSu9hHGq5o

- In Python we **never ever** use integer indeces to access object in a sequence (except arrays)
- Because it is error prone (of by one errors)
- People behind Python has taken care you do not need to

```
for color in colors:
    print color
```

YES



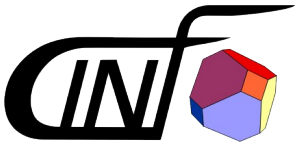
range(5) returns [0, 1, 2, 3, 4]

```
for index in range(len(colors)):
    print colors[index]
```

NO!!!

http://www.clipartbest.com/fireworks-drawing
http://www.clipartlord.com/category/military-clip-art/bomb-clip-art/

for loops the good way



- What if I need the index?
- Get the index along with the object

  enumerate()

- What if I have more than one sequence?
- Loop over several sequences at once

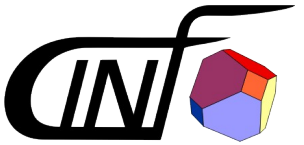
  zip()

Remember unpacking?

- Assign several value from a list to multiple variables at once

```
>>> mylist = [47, -1]
>>> value1, value2 = mylist
>>> value1
47
>>> value2
-1
```

enumerate – enumerate objects

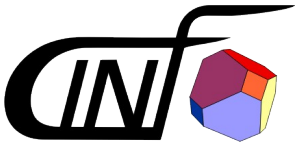


- Returns index-object pairs
- Can only be used directly in for loops
  - For use else-where call list on it: `list(enumerate(...))`

Notice un-packing in a for loop

```
>>> for index, color in enumerate(colors):
...     print 'Color ' + str(index) + ' is ' + color
...
Color 0 is magenta
Color 1 is cyan
```

zip – pair objects in sequences

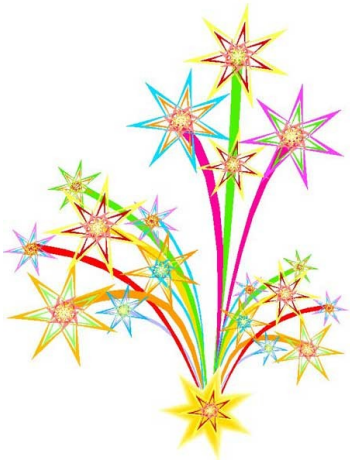


- Turns a pair of sequences into a sequence of pairs
- Can also only be used directly in for loops

```
>>> colors = ['magenta', 'cyan', 'crimson']
>>> shapes = ['square', 'circle', 'triangle']
>>> for color, shape in zip(colors, shapes):
...     print color, shape
...
magenta square
cyan circle
crimson triangle
```

You are now a native

- Congratulations, you are now a Python native
- That is 90% of what you need to know, to *loop like a native*
- Enjoy!

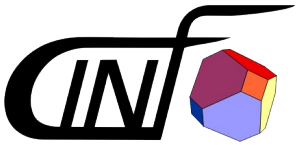


http://www.clipartbest.com/fireworks-drawing

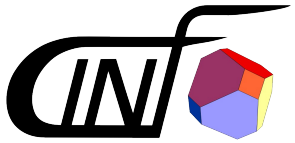
Loop like a native – Type along

Type along session in the terminal:
Loop like a native

Loop like a native – Exercises **IMPORTANT!!!**



- Make a list of measurement descriptions e.g: ['C1s', 'O1s']
- Make a list of measurment ids e.g: [47, 147]
- Loop over both the lists and print a plot legend out by concatenating them e.g: 'C1s – 47'
- Instead of printing them, add them to a new list of legends
- Using enumerate and this new list, print out a new set of legends in which they are enumerated e.g: 'Plot 0: C1s – 47'
- Experiment with using three lists in zip
- Experiment with using 2 lists in zip, but where they do not have the same length. What happens?



Center for Individual Nanoparticle Functionality

- What is a computer program
- Math
- Types
- Variables
- Containers
- Loops
- Boolean expressions
- Control flow (if-elif-else)
- Functions

Python part 0

Boolean expression

- An expression that evaluates to True of False
- Used to make checks and perform decisions
- Simple (binary) operators are:

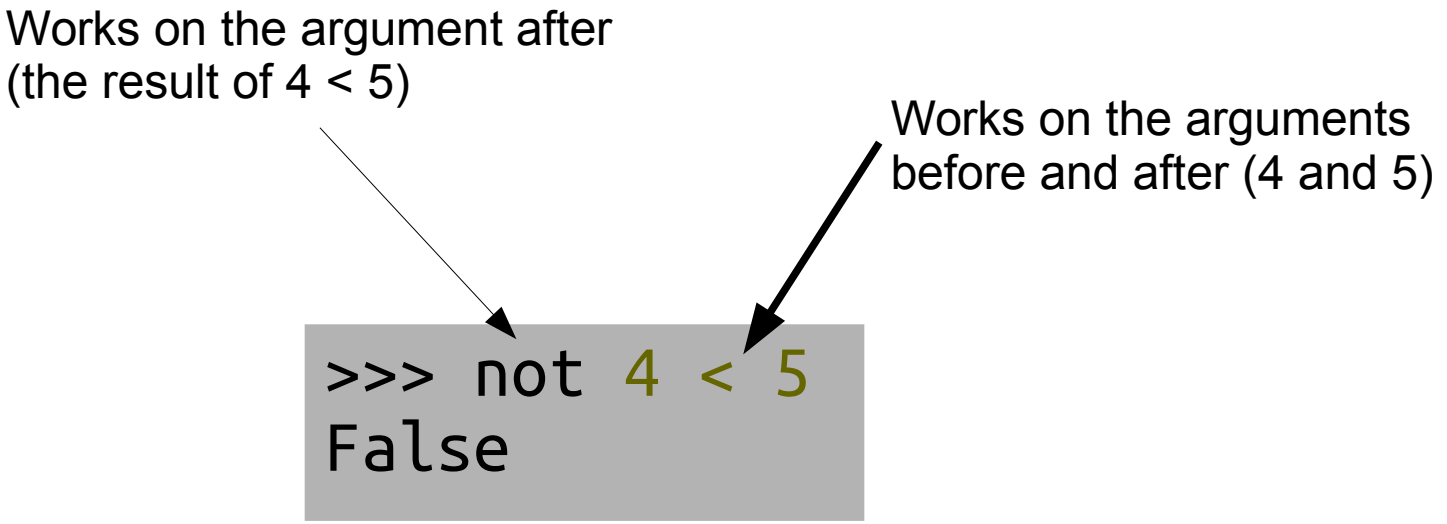
Operator	Explanation	Example	Result
>	greater than	5 > 5	False
>=	greater than or equal	5 >= 5	True
<	smaller than	-3 < 10	True
<=	smaller than or equal	47 <= 1	False
==	equals (carefull with floats)	10 == 10	True
!=	not equals	10 != 10	False
in	in (a sequence)	4 in [1, 2, 4, 7]	True

Boolean expressions

```
>>> 4 < 5
True
>>>
>>> 5 != 10
True
>>> 5 != 10
True
>>>
>>> 7 in [1, 2, 3]
False
>>> 3 in [1, 2, 3]
True
```

Special operator **not**

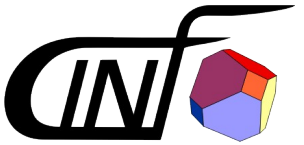
- **not** is a single argument operator that negates the following expression
- Special syntax for **in** (and is) for readability
- Also works with !=



not

```
>>> 4 < 5
True
>>> not 4 < 5
False

>>> 4 in [2, 4, 6]
True
>>> not 4 in [2, 4, 6]
False
>>> 4 not in [2, 4, 6]  # Also works
False
```

Putting boolean expression together



- Boolean axpression can be put together to create more complex expressions for checks or decisions
- Binary operators **and** and **or**
- **and** - True when both are True
- **or** – True when at least one is True

Truth tables

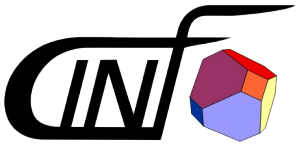
and

A	B	Result
True	True	True
True	False	False
False	True	False
False	False	False

or

A	B	Result
True	True	True
True	False	True
False	True	True
False	False	False

Putting boolean expressions together



```
>>> 4 < 5 and 1 < 2
True
>>> 4 < 5 and 1 > 2
False
>>> 4 < 5 and not 1 > 2
True
>>> 4 < 5 and 2 in [2, 3]
True
```

Boolean expressions – Type along

Type along session in the terminal:
Boolean expressions

Boolean expressions – Exercises

- Make an expression that checks if a number is in range i.e. between two others
- Make two sets of metadata in a dict e.g:

  {'label': 'C1s', 'monochromator': True}

  {'label': 'O1s', 'monochromator': False}
- While testing on both
  - Make an expression that checks whether it is a C1s measurement
  - Make an expression that checks whether it is monochromated
  - Make an expression that checks for both conditions
  - Make an expression that checks for either
  - Make an expression that check fo C1s and NOT monochromated

- What is a computer program
- Math
- Types
- Variables
- Containers
- Loops
- Boolean expressions
- Control flow (if-elif-else)
- Functions

Python part 0

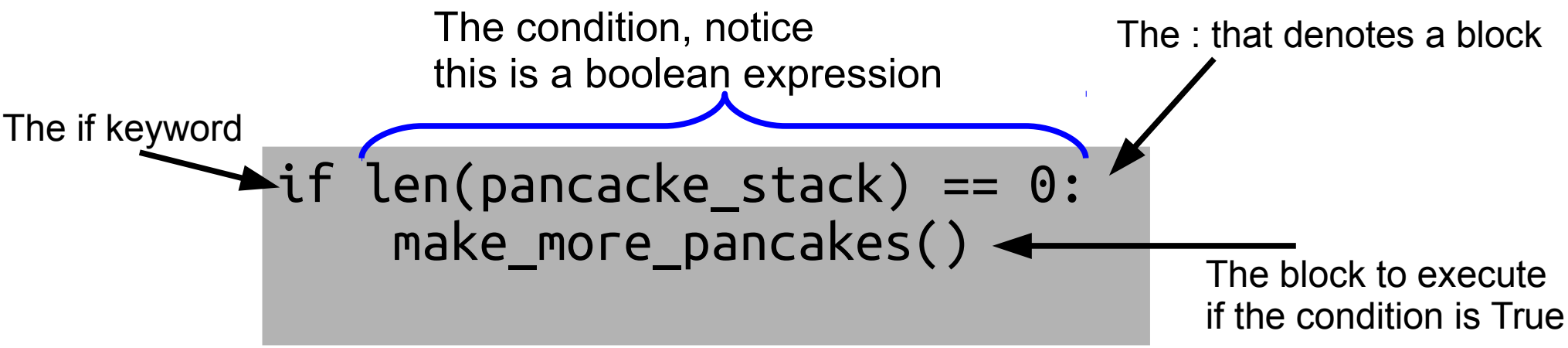
Control flow (if-elif-else)

- Used to make decisions
- Code is executed from the top down
- A decision represents a cross-roads
- Execute only certain lines of code depending on a boolean expression
  - Aha! That is what they are for
- Decisions are very fundamental to all programming

http://www.123rf.com/clipart-vector/intersection.html

The **if** statement

- The simple **if** statement is used to optionally execute something
- Think “If these conditions are met, also do this”
- “If pan-cake plate is empty, make more”



The **if-else** structure

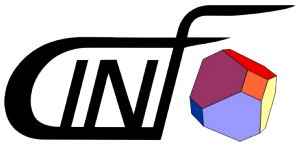
- Do something if and in all other cases do something else
- Think “If this is the case, do this, and if not, do something else”
- Classic cross-roads
- Always execute one block or the other

```
if len(pancacke_stack) == 0:
    make_more_pancakes()
else:
    keep_eating()
```

Notice: No condition after an else statement

The else block

The **if-elif-elif-else** structure



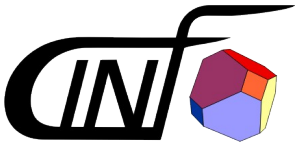
- **elif** is short for “else if”
- Adds the option, to one or more extra specific conditions to the decision, before the default **else**

```
if len(pancacke_stack) == 0:
    make_more_pancakes()
elif len(pancake_stack) == 1:
    heat_up_pan()
else:
    keep_eating()
```

Note: The elif also has an condition

The **elif** block

The complete **if-elif-else** structure



- Note, the else is optional, also after elif
- The complete allowed structure is:
  - 1 **if**
  - Followed by 0 or more **elif**
  - Followed by 0 or 1 **else**

if-elif-else – Type along

Type along session in the terminal:
if-elif-else

if-elif-else – Exercises

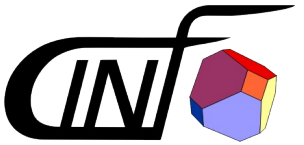
- Make a structure that checks a number and optionally prints out a message if the number is <0
- Make a structure that checks if there is content in a list and prints out two different message is there is and if there is not
- Make a structure that cheks an age, <13(child), >65(senior) and in between (adult) and prints out an appropriate greeting for each age group
- Make a structure that checks if a number is in range, is it if prints out the number and if not prints out a warning

Center for Individual Nanoparticle Functionality

- What is a computer program
- Math
- Types
- Variables
- Containers
- Loops
- Boolean expressions
- Control flow (if-elif-else)
- Functions

Python part 0

Functions

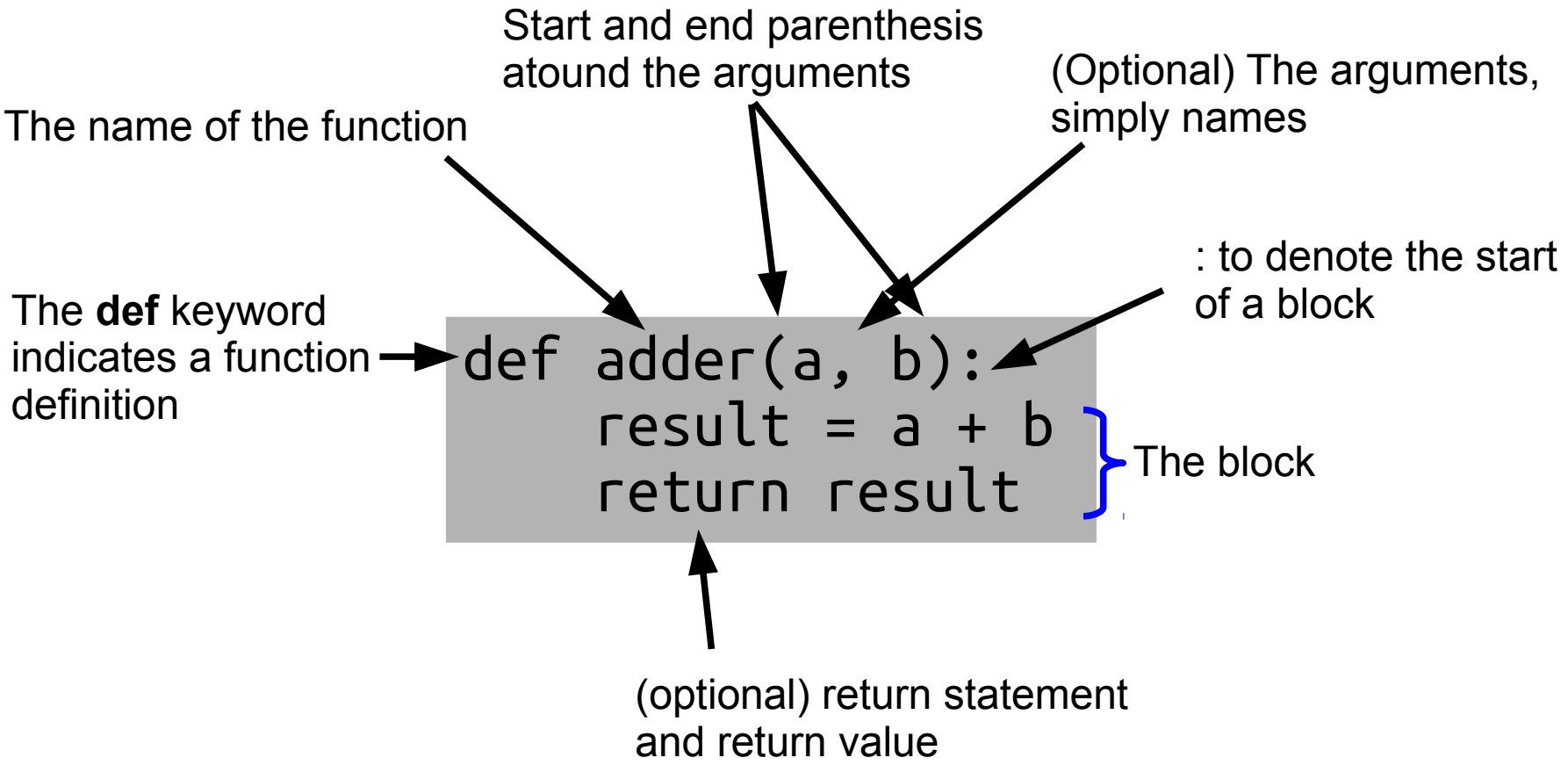


- A group of lines with well-defined (optional) input and (optional) output
- Used to:
  - **Re-use certain pieces of code**
  - **Organisation, to break up code into logical parts**
- *Very fundamental* to the way programs are structured
- Think “Functions perform a well defined job”
- This is where programming starts to get pratical

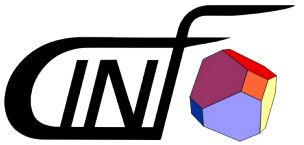
The anatomy of a function

```
def adder(a, b):
    result = a + b
    return result
```

The anatomy of a function



More on functions



- Variables defined inside functions, including arguments, exist only inside the function
- The arguments are optional, but the () are not
- The return statement is optional, if missing the function will return None
  - Remember **None** means “nothing to see here”

Arguments

- Arguments are used in order

```
def adder(a, b):
    result = a + b
    return result

adder(1, 2)

# a will be 1 and b will be 2
```

Function examples

- Perform task, no return

```
def print_metadata(metadata):
    print 'Title: ' + metadata['title']
    print 'Id: ' + str(metadata['id'])
    print 'Comment: ' + metadata['comment']

my_meta = {'title': 'C1s', 'id': 47, 'comment': 'This is the one'}

print_metadata(my_meta)
```

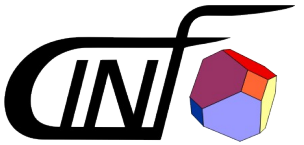
Function examples

- Perform task, with return

```
def make_metadata_string(metadata):
    result = 'Title: "' + metadata['title']
    result = result + '" Id: ' + str(metadata['id'])
    result = result + ' Comment: "' + metadata['comment'] + '"'
    return result

my_meta = {'title': 'C1s', 'id': 47, 'comment': 'This is the one'}

metadata_string = make_metadata_string(my_meta)
print metadata_string

# Output
Title: "C1s" Id: 47 Comment: "This is the one"
```

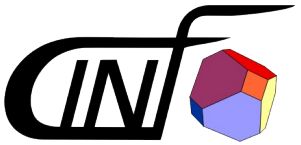
There is a better way to format numbers into strings, we will talk about that in part1

Function can call other functions



- Function can call other functions
- This is the way you make structure when complexity of you task increases
- They can also call themselves (recursion), but be carefull, that can produce infinite loops

Function can call other functions

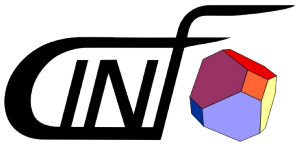


```
def perform_subtask0():
    print 'Hello'

def perform_subtask1():
    print 'world'

def perform_task():
    perform_subtask0()
    perform_subtask1()

perform_task()
```

functions – Type along

Type along session in the terminal:
functions

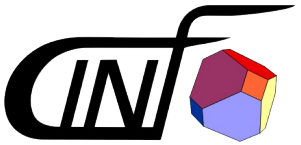
functions – Exercises



- Write a function “calculate” that performs a calculation with several parameters and call it
- Write a function “do_calculate” that calls the “calculate” and call it
- Write a function that returns the sum of every other element in a list
- Write a function that, given a list, returns a new list where all objects are multiplied by 2

Putting it all together

Putting it all together



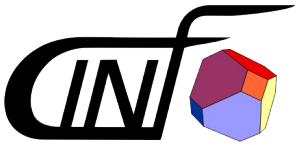
```
# ... get_data and get_normalization not show

def analyze_data_set(data_set, normalization):
    # Fancy analysis
    return result

def analyze_data_sets(data_sets):
    normalization = get_normalization(data_sets)
    results = []
    for data_set in data_sets:
        result = analyze_data_set(data_set, normalization)
        results.append(result)
    return results

def main():
    data_sets = get_data()
    results = analyze_data_sets(data_sets)
    # plot data sets and results

main()
```

Center for Individual Nanoparticle Functionality

- Summary

Python part 0

Scientists Summary



- Math: Take care with division
- Objects have a type that defines what they can do
- Variables are names for objects
- Lists and dicts are the bread and butter of containers
- Use for and while loops for repetition
  - And loop like a native
- Boolean expressions, put together **and**, **or**
- If-elif-else to make decisisions
- Functions for calculation, organisation and code re-use

Homework

- Getting good at a new syntax (e.g. Pythons) has a lot to do with muscle memory

  Start writing Python and do it today

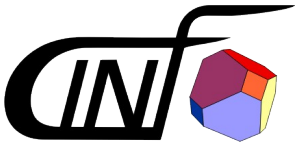
- Work through this tutorial:

  https://openhatch.org/wiki/Boston_Python_Workshop_8/Friday/Tutorial

  (and do actually type all the examples), for the repetition and for getting to type some more

- The link above is also to a large part the inspiration to this presentation

More optional homework

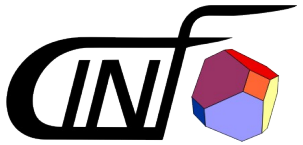


- http://codingbat.com/python

  Elementary exercise that you can run on the website. Good for that muscle memory.

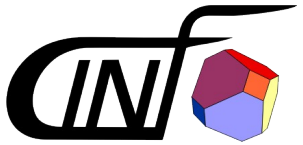
- http://learnpythonthehardway.org/book/

  (Don't mind the comments about using a specific environment)



Have fun and see you at part 1

THE END



Extra slides, won't need them

Extras

- abs()
- min()
- max()
- bool()
- any()
- all()
- is
- id()
- sum()
- round()