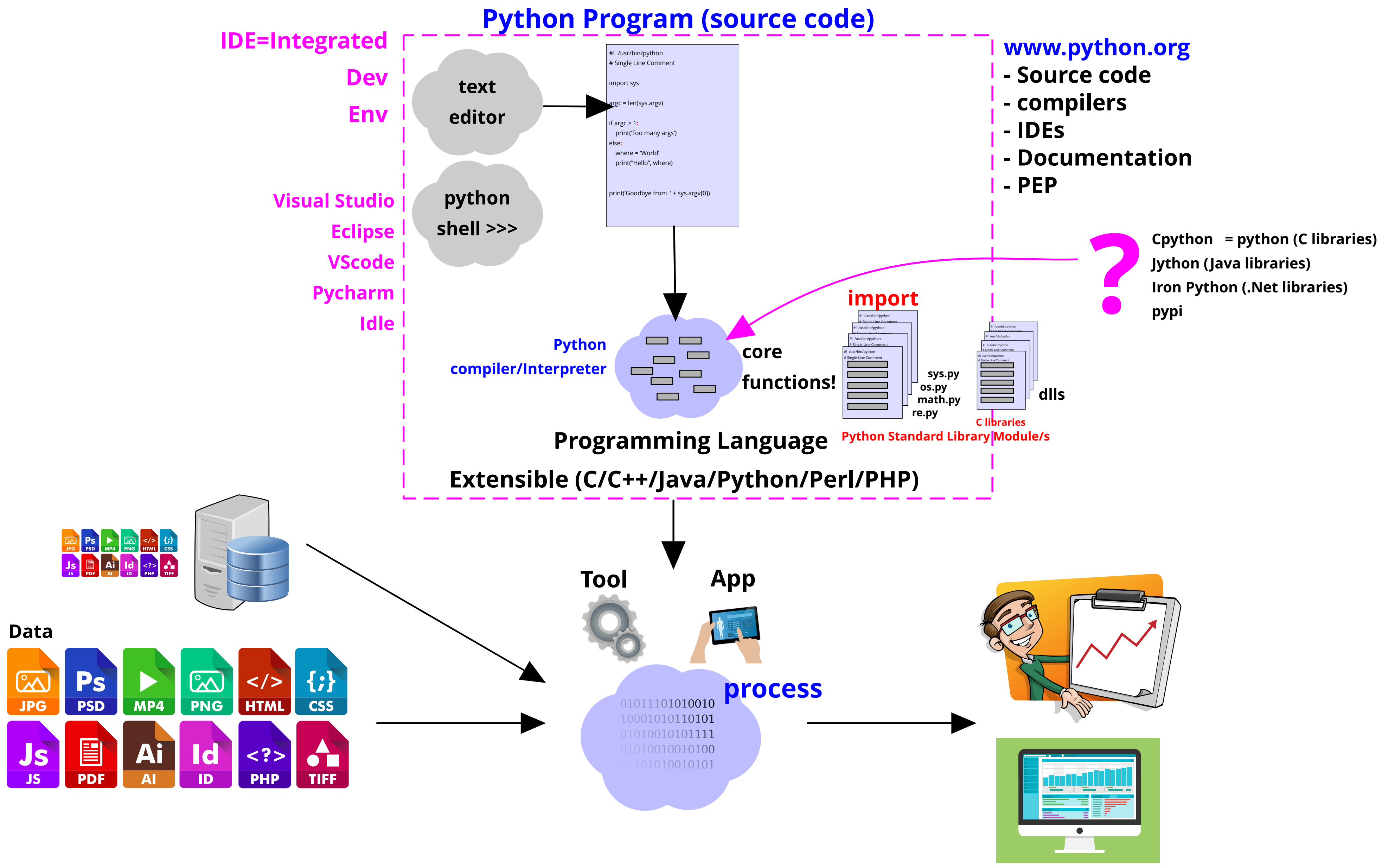
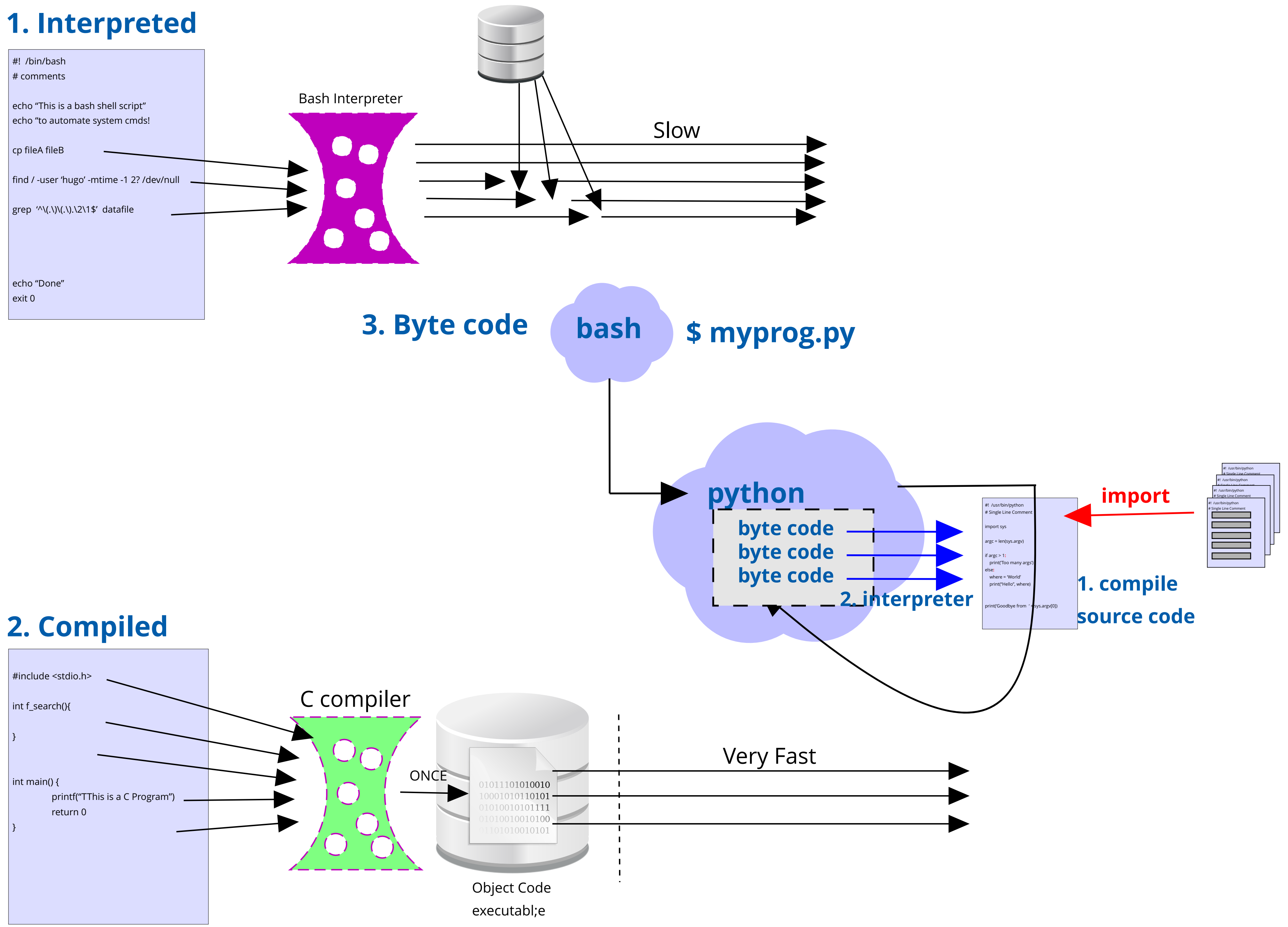
Ch2: Anatomy of a Python Program



```
#!  /usr/bin/python
# Single Line Comment

import sys

argc = len(sys.argv)

if argc > 1:
    print('Too many args')
else:
    where = 'World'
    print("Hello", where)


print('Goodbye from  ' + sys.argv[0])
```

Ch2: Compiled & Interpreted Languages



1. Interpreted

```
#!  /bin/bash
# comments

echo “This is a bash shell script”
echo “to automate system cmds!

cp fileA fileB

find / -user ‘hugo’ -mtime -1 2? /dev/null

grep  ‘^\(.\)\(.\).\2\1$’  datafile



echo “Done”
exit 0
```

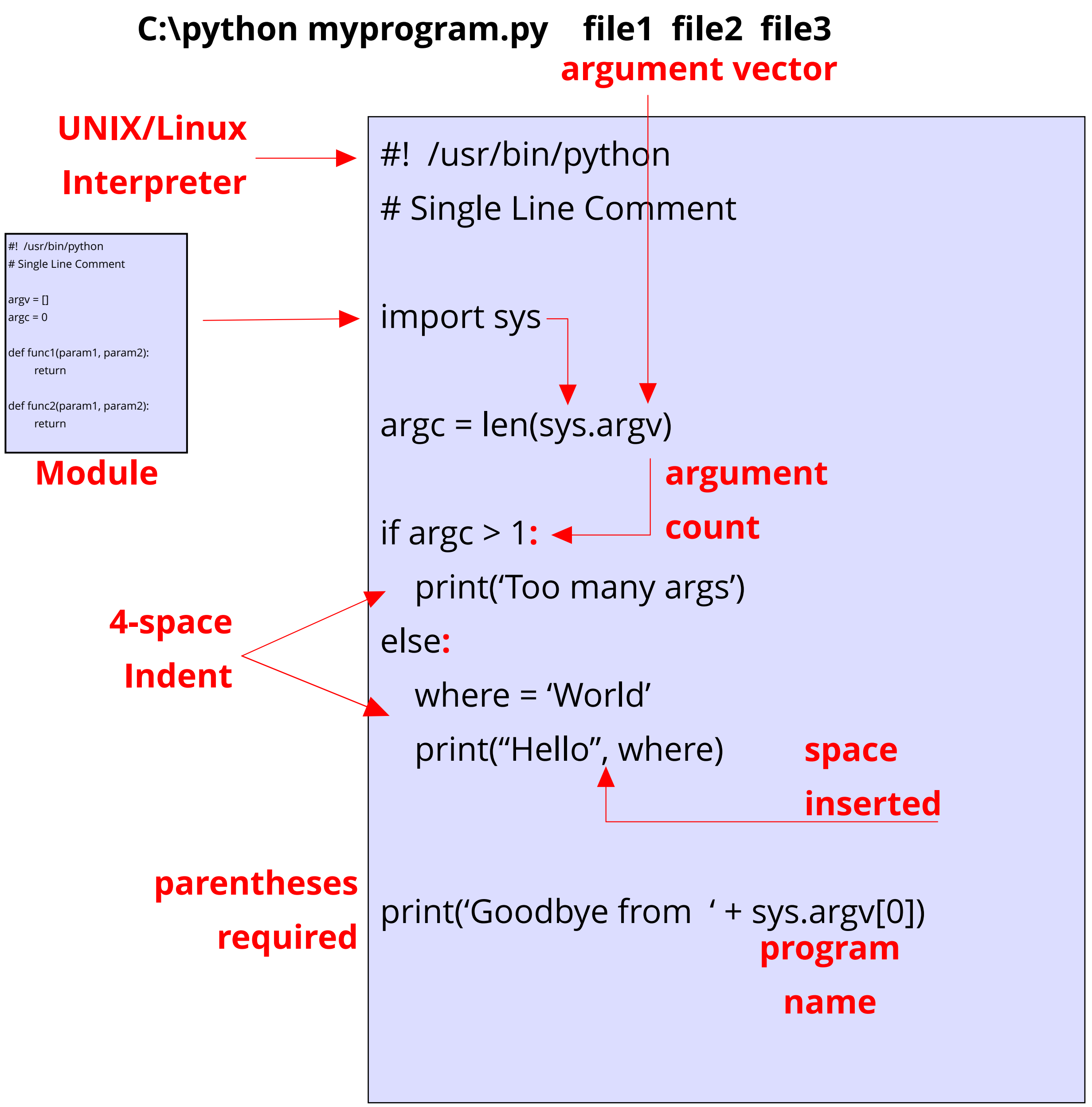
3. Byte code

```
#!  /usr/bin/python
# Single Line Comment

import sys

argc = len(sys.argv)

if argc > 1:
   print(‘Too many args’)
else:
   where = ‘World’
   print(“Hello”, where)


print(‘Goodbye from  ‘ + sys.argv[0])
```

2. Compiled

```
#include <stdio.h>

int f_search(){

}


int main() {
          printf(“TThis is a C Program”)
          return 0
}
```

Ch2: Anatomy of a Python Program

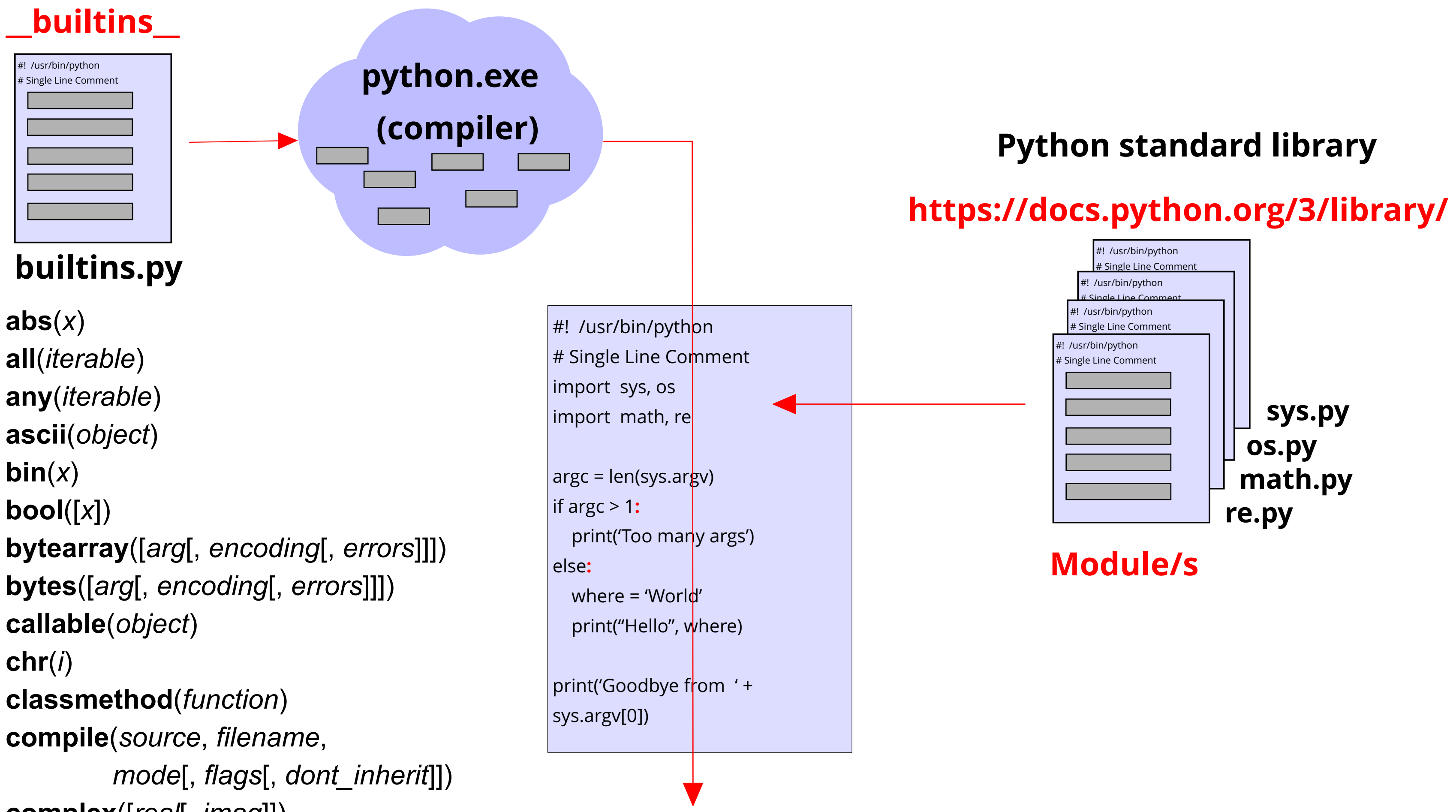


```
C:\python myprogram.py    file1  file2  file3
```

```
#!  /usr/bin/python
# Single Line Comment
argv = []
argc = 0
def func1(param1, param2):
        return
def func2(param1, param2):
        return
```

```
#!  /usr/bin/python
# Single Line Comment


import sys


argc = len(sys.argv)


if argc > 1:
   print('Too many args')
else:
   where = 'World'
   print("Hello", where)


print('Goodbye from  ' + sys.argv[0])
```

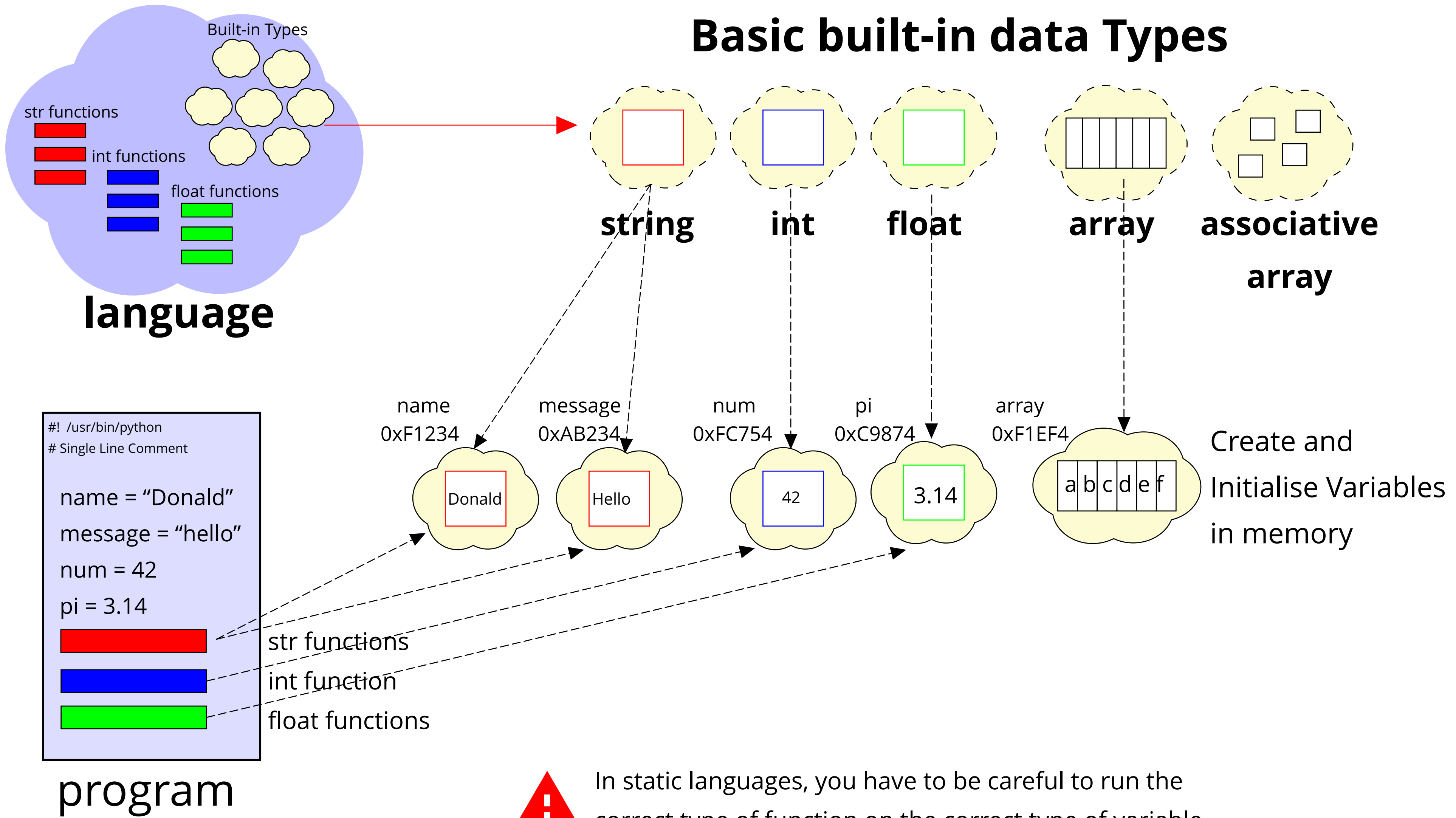
Ch2: Python Modules



```
#!  /usr/bin/python
# Single Line Comment
import  sys, os
import  math, re

argc = len(sys.argv)
if argc > 1:
   print('Too many args')
else:
   where = 'World'
   print("Hello", where)

print('Goodbye from  ' +
sys.argv[0])
```

abs(*x*)
all(*iterable*)
any(*iterable*)
ascii(*object*)
bin(*x*)
bool([*x*])
bytearray([*arg*[, *encoding*[, *errors*]]])
bytes([*arg*[, *encoding*[, *errors*]]])
callable(*object*)
chr(*i*)
classmethod(*function*)
compile(*source*, *filename*,
mode[, *flags*[, *dont_inherit*]])
complex([*real*[, *imag*]])
delattr(*object*, *name*)
dict([*arg*])
dir([*object*])
divmod(*a*, *b*)
iter(*o*[, *sentinel*])
len(*s*)
list([*iterable*])
locals()

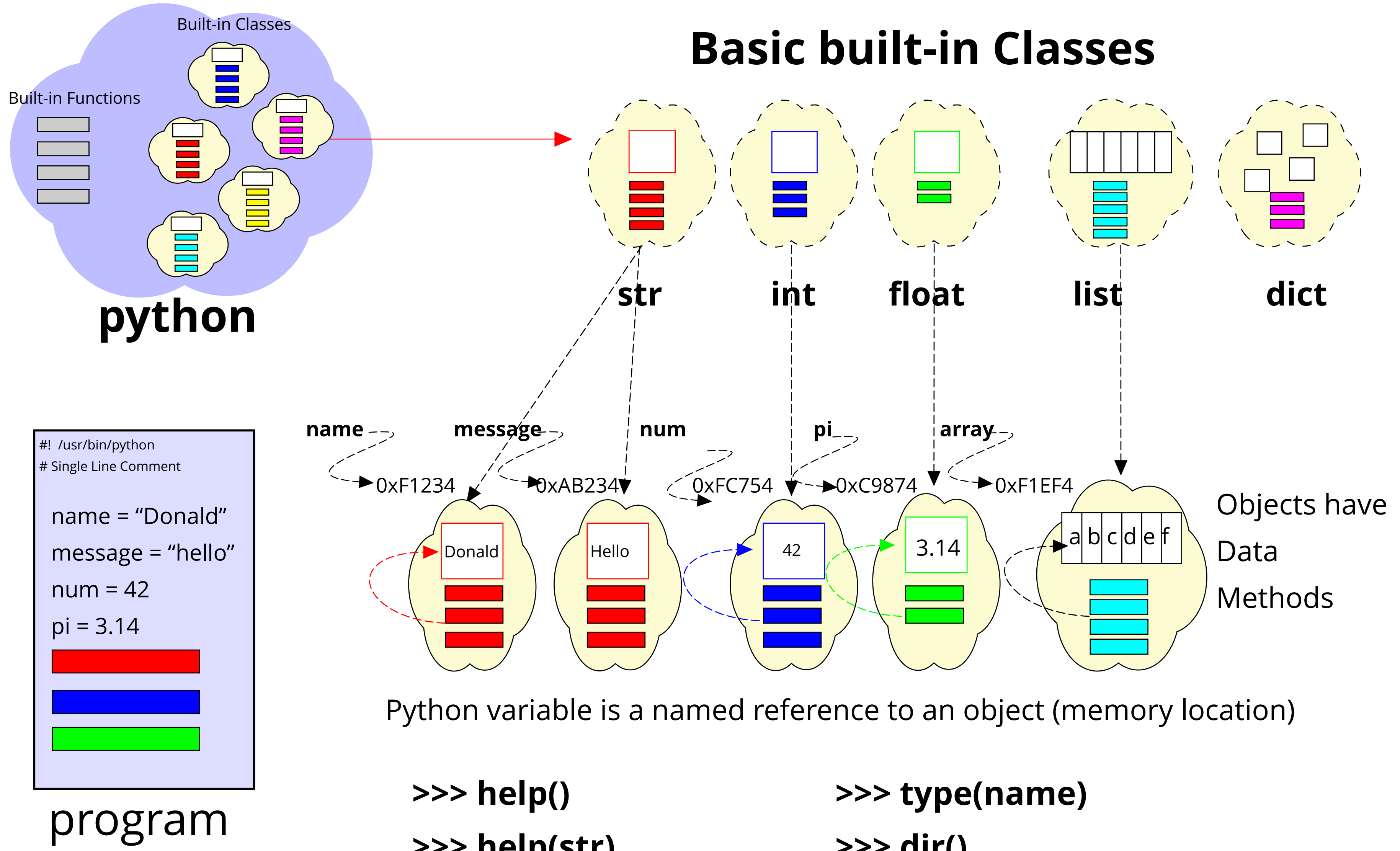
Ch2: Fundamental Variables



```
#!  /usr/bin/python
# Single Line Comment

name = “Donald”
message = “hello”
num = 42
pi = 3.14
```

In static languages, you have to be careful to run the correct type of function on the correct type of variable

Ch2: Objects (variables)



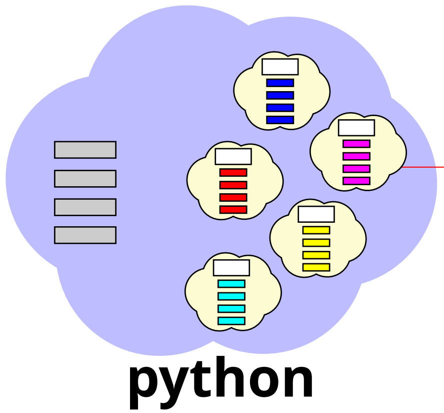
```
#!  /usr/bin/python
# Single Line Comment

name = “Donald”
message = “hello”
num = 42
pi = 3.14
```

Objects have Data Methods

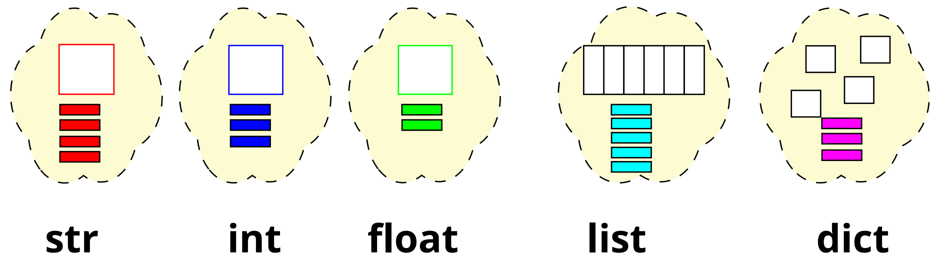
Python variable is a named reference to an object (memory location)

```
>>> help()                >>> type(name)
>>> help(str)             >>> dir()
>>> help(int)             >>> dir(str)
>>> help()                >>> dir(list)
>>> help(str.lower)       >>> dir(name)
```

Ch2: Objects (variables)



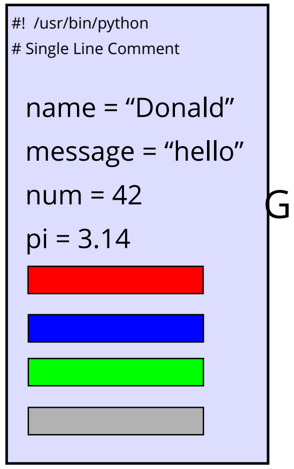
Basic built-in classes



Python is CaSe sensitive

CASE CONVENTION (Style Guide)

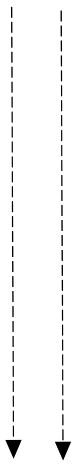
taxrate	lowercase
TaxRate	ProperCase
taxRate	camelCase
tax_rate	snake_case

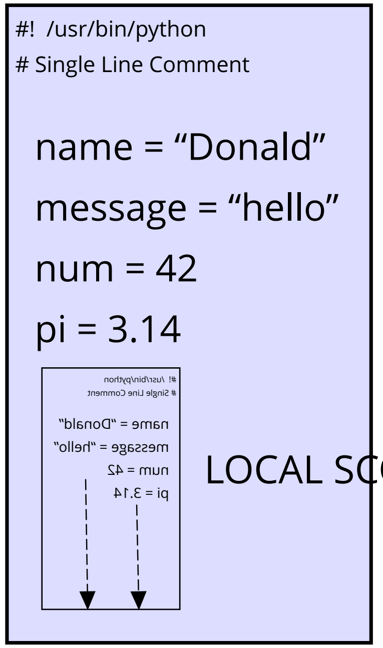


```
#!  /usr/bin/python
# Single Line Comment

name = "Donald"
message = "hello"
num = 42
pi = 3.14
```

program

GLOBAL SCOPE





```
#!  /usr/bin/python
# Single Line Comment

name = "Donald"
message = "hello"
num = 42
pi = 3.14
```

globals()

locals()

LOCAL SCOPED VARIABLE/OBJECT

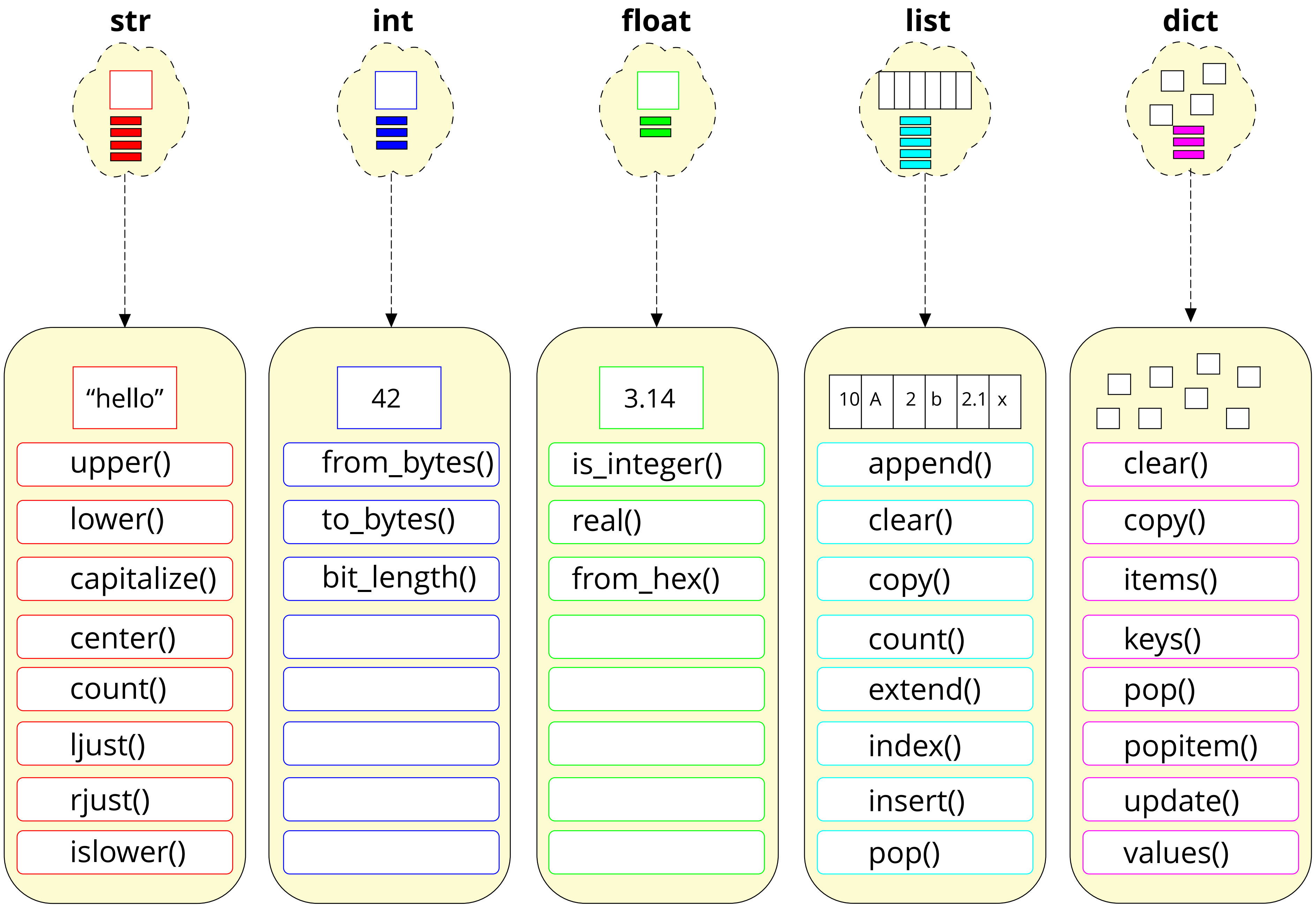
1. Life
2. Visibility

Scope

_myvar	private(module/class)
__myvar	mangled(protected)
__myvar__	(special)

Ch2: Methods - exercise

```
>>> dir(str)
```



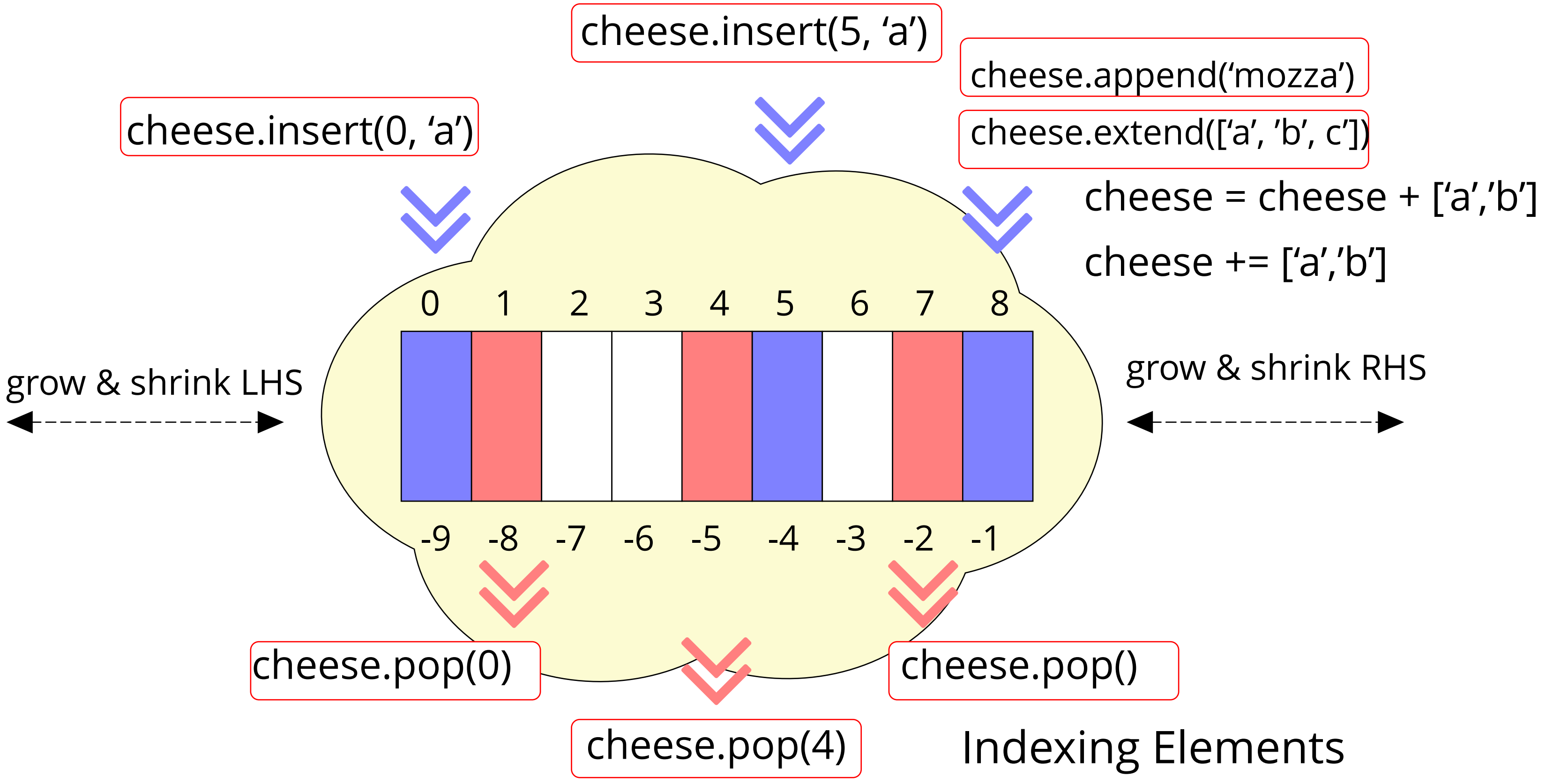
str	int	float	list	dict
"hello"	42	3.14	10 | A | 2 | b | 2.1 | x	
upper()	from_bytes()	is_integer()	append()	clear()
lower()	to_bytes()	real()	clear()	copy()
capitalize()	bit_length()	from_hex()	copy()	items()
center()			count()	keys()
count()			extend()	pop()
ljust()			index()	popitem()
rjust()			insert()	update()
islower()			pop()	values()

Fill in the names of methods for each object (class)

Ch2: Python Lists

Ordered sequence, mutable, dynamic in size

```
>>> cheese = ['cheddar', 'stilton', 'edam', 'gorgonzola']
>>> print(cheese[2])                          'edam'
>>> print(cheese[-1])                         'gorgonzola'
```



Searching: cheese.index('stilton')

Sorting: cheese.sort()

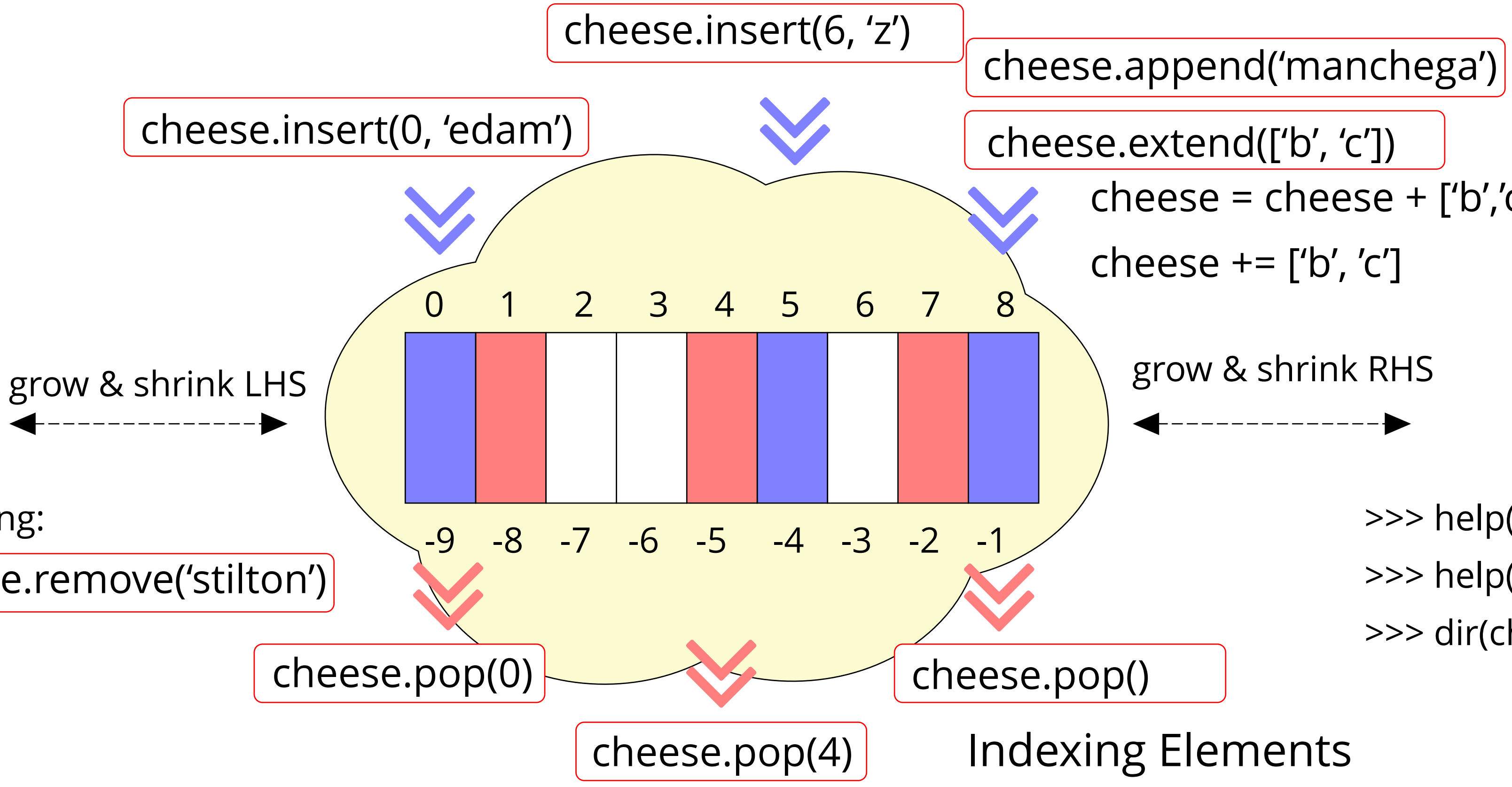
Indexing Elements

cheese[1:3]	- element 1 to 2 (but not 3)
cheese[0:]	- all the elements
cheese[-1]	- last element

Ch2: Python Lists

Ordered sequence, mutable, dynamic in size

```
>>> cheese = ['cheddar', 'stilton', 'edam', 'gorgonzola']
>>> print(cheese[2])                                'edam'
>>> print(cheese[-1])                               'gorgonzola'
```



Removing:

`cheese.remove('stilton')`

```
>>> help(list)
>>> help(list.sort)
>>> dir(cheese)
```

Searching: `cheese.index('edam')`

Sorting: `cheese.sort()`

Reverse: `cheese.reverse()`

Indexing Elements

cheese[1:3] - element 1 to 2 (but not 3)

cheese[0:] - all the elements

cheese[-1] - last element

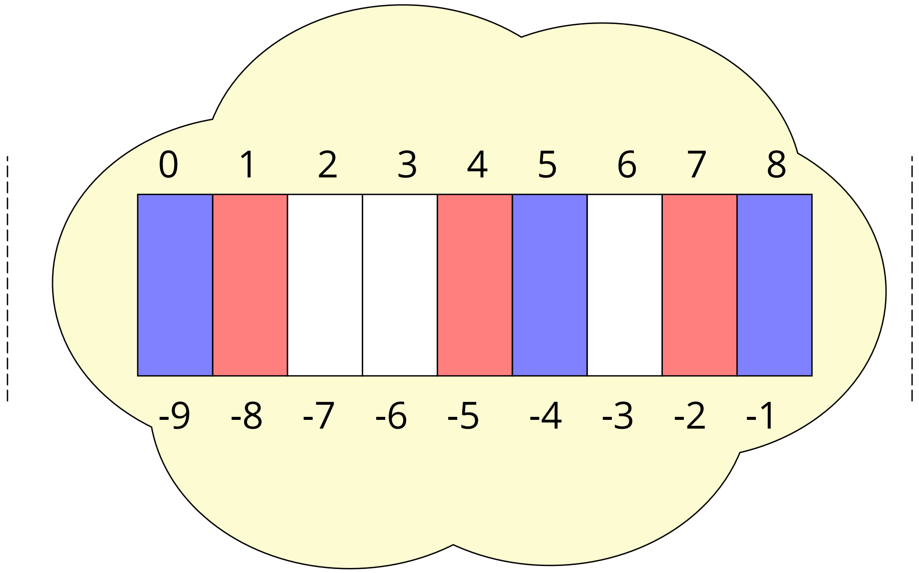
Ch2: Python Tuples

Ordered sequence, immutable

```
>>> mytuple = 'eggs', 'bacon', 'spam', 'tea'
```

help(tuple)

dir(tuple)

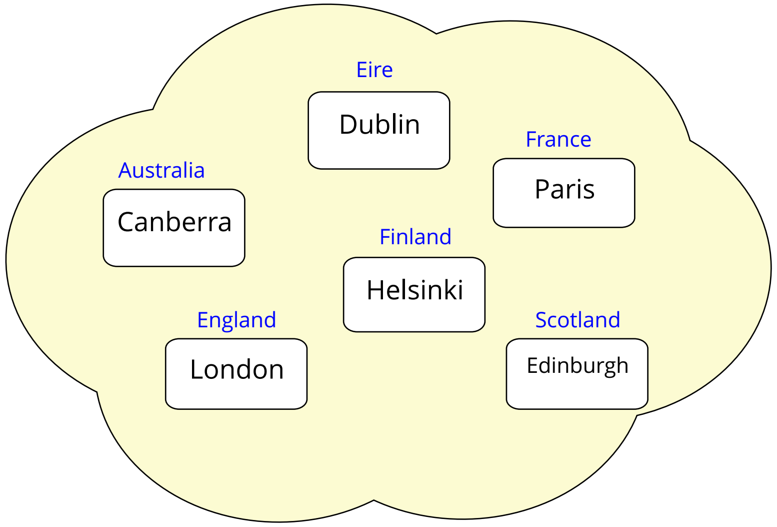


```
>>> print(mytuple[1])          'bacon'
>>> print(mytuple[-1])         'tea'
>>> mytuple[2] = 'muffin'      ✗
```

Ch2: Python Dictionaries (hashes, hash tables, associative arrays)

Unordered sequence, mutable and dynamic in size, faster searching

```
>>> capitals = {‘Australia’:‘Canberra’, ‘Eire’:‘Dublin’,
                ‘France’:’Paris’, ‘Finland’:’Helsinki’,
                ‘England’:’London’, ‘Scotland’:’Edinburgh’}
```

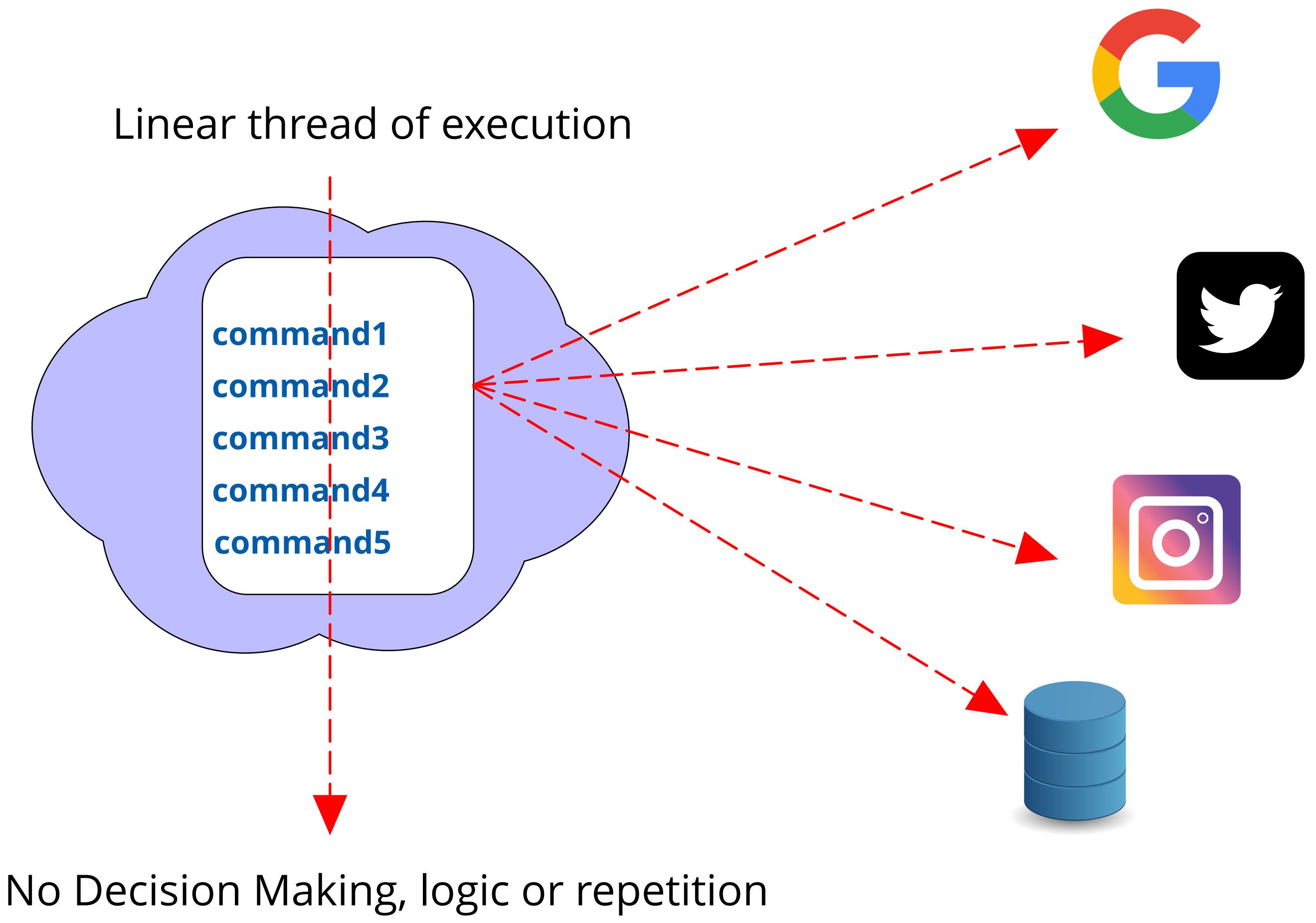


```
>>> capitals[‘Germany’] = ‘Berlin’
>>> print(capitals[‘France’])
```

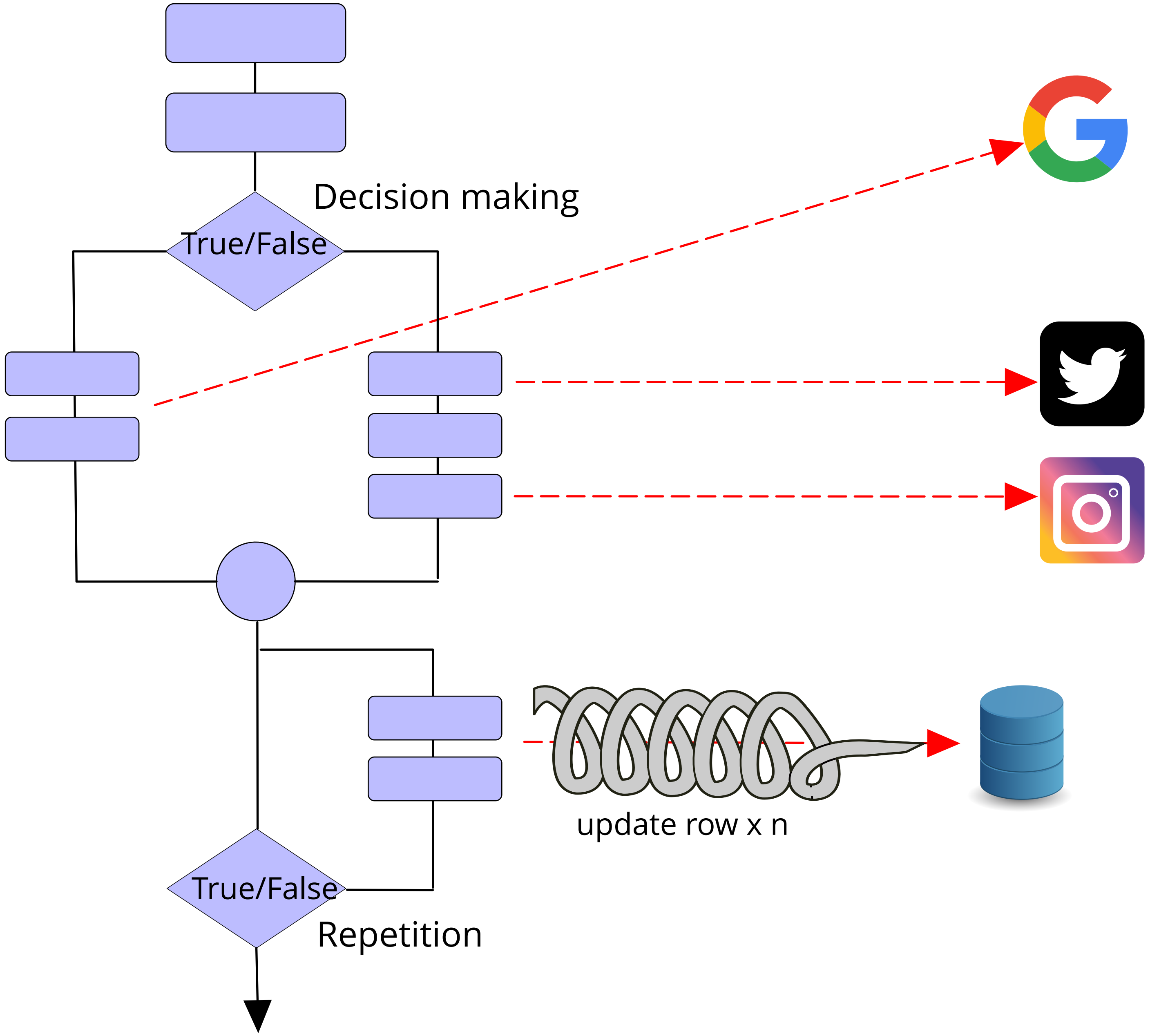
Use PEP 008 guidelines when writing GOOD Python code!

Details layout of code including space, tabs, indentations etc

Ch3: Flow Control

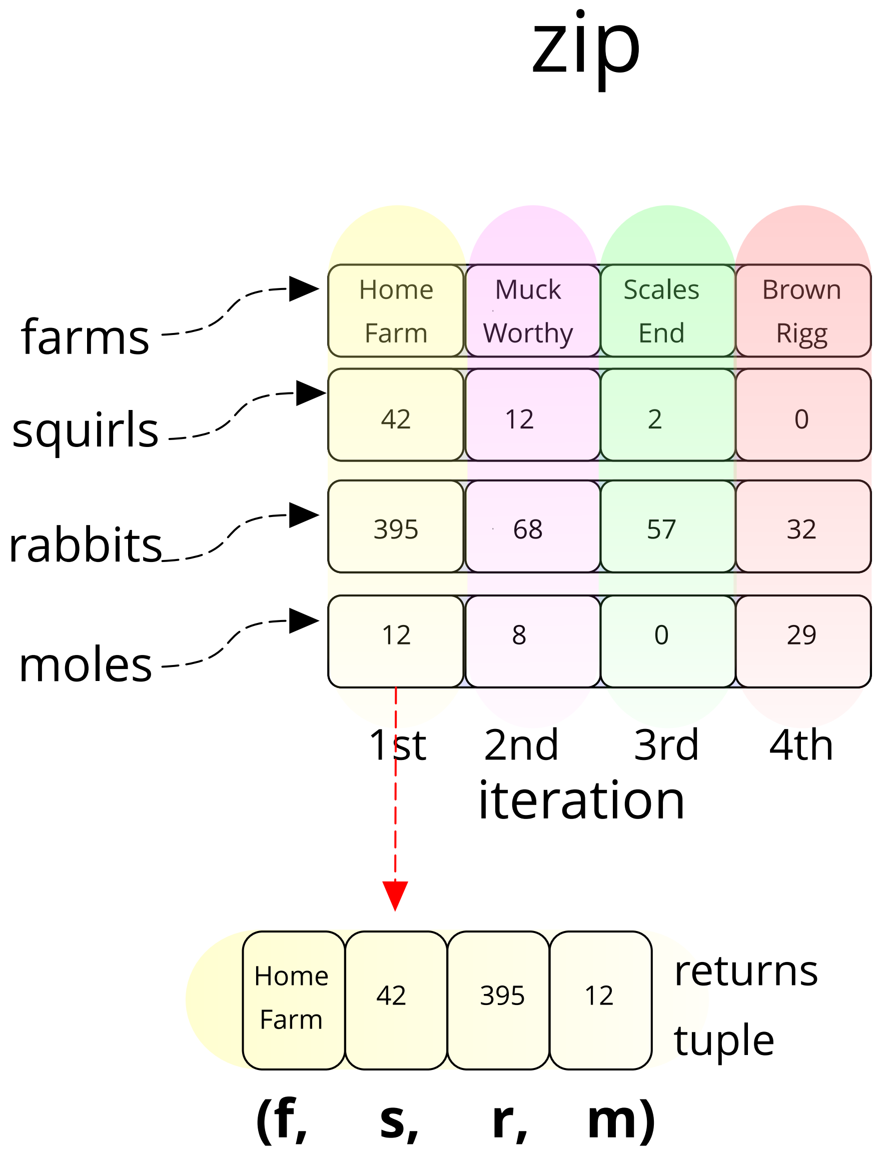

Linear thread of execution
command1
command2
command3
command4
command5
No Decision Making, logic or repetition

Ch3: Flow Control



Ch3: Zipping through multiple lists

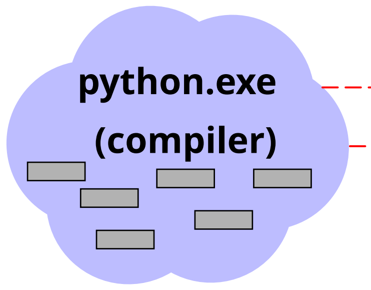
```
#!  /usr/bin/python
# Single Line Comment

farms = ['Home Farm', 'Muckworthy',
            'Scales End', 'Brown Rigg']
squirls = [42, 12, 2, 0]
rabbits = [395, 68, 57, 32]
moles   = [12, 8, 0, 29]



for f, s, r, m in zip(farms, squirls, rabbits, moles):
    print ('Total for', f, ':', s + r + m)
```



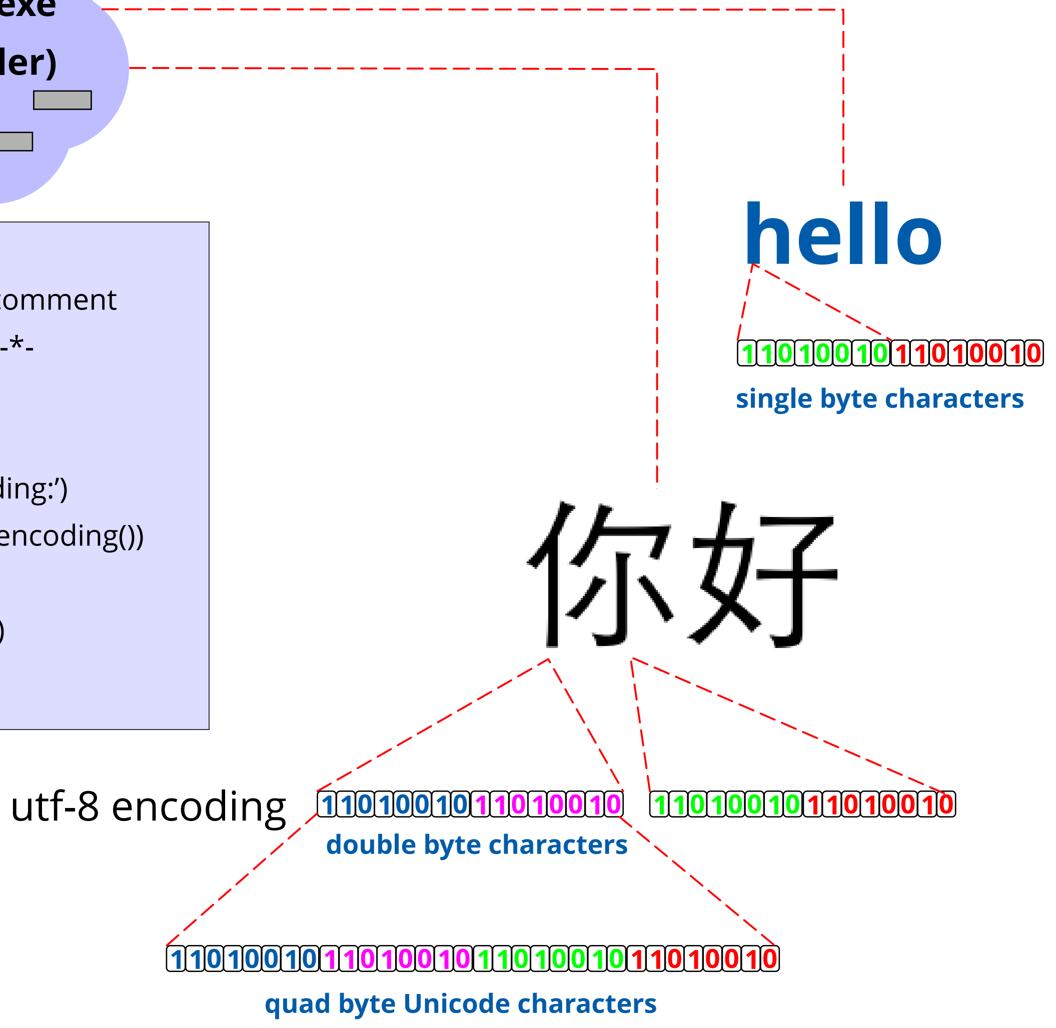
Py2 - zip returns list of tuples: [(tuple1), (tuple2), (tuple3), (tuple4)]

Py3 - zip returns iterator of tuples: give me next next (tuple) when I need it….

Ch4: Python 3 strings

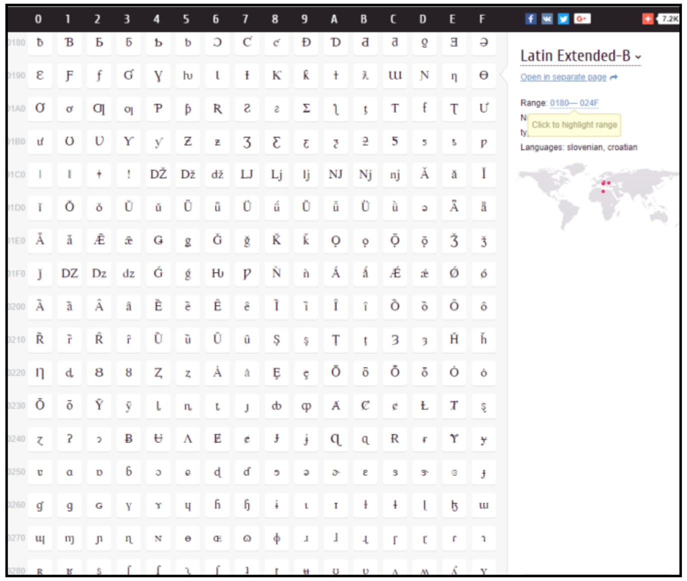


```
#!  /usr/bin/python
# Special PRAGMA comment
# -*- coding: latin-1 -*-

import sys
print('Current encoding:')
print(sys.getdefaultencoding())

print('\N{euro sign}')
print('\u20ac')
```



```
>>> euro = "\u20ac"
>>> print(euro)                €
>>> euro = "\N{euro sign}"
>>> print(euro)                €

>>> str.encode("hello", "utf-8")
b'hello'
>>> bytes.decode(b"hello", "utf-8")
'hello'
```

Unicode Char Set



Ch4: The print function

I think this
duck typed this!

positional arguments　　　　named arguments

Py3

```
print(object1, object2, object3, object4, object5, sep=" ", end="\n")
```

Can be any object class

int　*str*　*list*　*tuple*

```
print(myAge, myName, studentList, lottoNums, myTank)
```

Escape Chars

\n = newline
\t = tab
\v = vertical tab
\f = form feed
\N{euro sign} = Unicode char

```
print(r"c:\programs\newproject\test")
```

Ch4: Nice print - where's printf?

```
#!  /usr/bin/python

planets = {'Mercury' : 57.91,
              'Venus'   : 108.2,
              ' Earth'   : 149.597870,
               'Mars'    : 227.94}

for i, key in enumerate(planets.keys(), 1):
   print("{:2d} {:<10s} {:06.2f} Gm". format(i, key, planets[key]))
```

```
1 Earth      149.60 Gm
2 Mercury    057.91 Gm
3 Mars       227.94 Gm
4 Venus      108.20 Gm
```

```
text = 'hello'
print(text.capitalize())
print(text.upper())
print('<'+text.center(12)+'>')
print('<'+text.ljust(12)+'>')
print('<'+text.rjust(12)+'>')
print('<'+text.zfill(12)+'>')
```

```
Hello
HELLO
<   hello    >
<hello       >
<       hello>
<0000000hello>
```

```
for i, key in enumerate(planets.keys(), 1):
   print(f"{i:2d} {key:<10s} {planets[key]:06.2f} Gm")
```

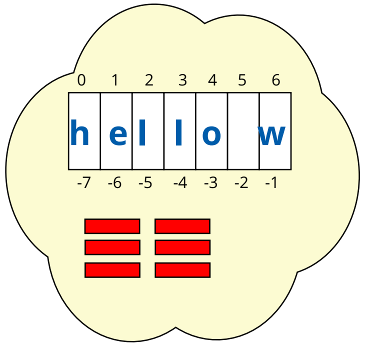
```
1 Earth      149.60 Gm
2 Mercury    057.91 Gm
3 Mars       227.94 Gm
4 Venus      108.20 Gm
```

"{0:2d} {1:<10s} {2:06.2f} Gm". format(i, key, planets[key])

f-string =   f"{i:2d} {key:<10s} {planets[key]:06.2f} Gm" ☺

Ch5: Collections - Ordered Sequences

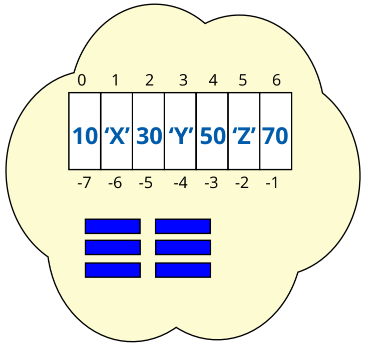
str



Ordered
Indexed using [n]
Immutable

```
message = 'hello'
message.upper()
message[0:2]
message[-1]
```

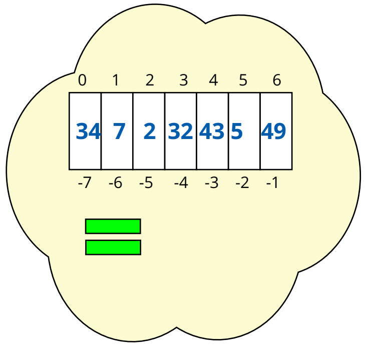
list



Ordered
Indexed using [n]
Mutable
Dynamic, flexible

```
mylist = [10,'x',20,'y']
mylist.append(30)
mylist.insert(0,'a')
mylist[0:3]
mylist[-1]
```

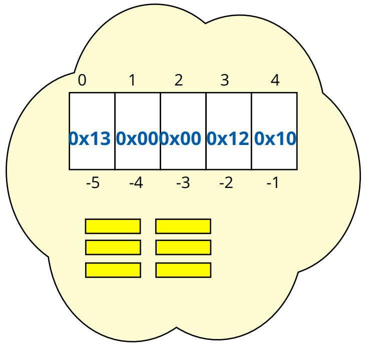
tuple



Ordered
Indexed using [n]
Immutable
Fast, simple

```
lotto = 37,7,2,32,45
lotto.count(7)
lotto.index(32)
lotto[2:4]
lotto[-1]
```

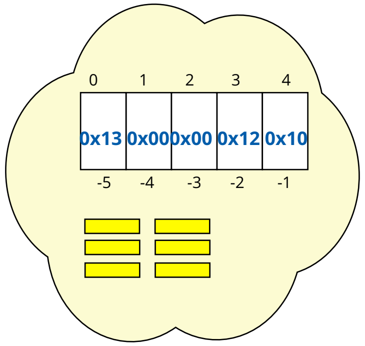
bytearray



Ordered
Indexed using [n]
Raw Binary Data(2.6)
Mutable

```
key = bytearray([0x12, 0x00, 0x13])
key.islower()
key.decode()
key[0:2]
key[-1]
```

bytes

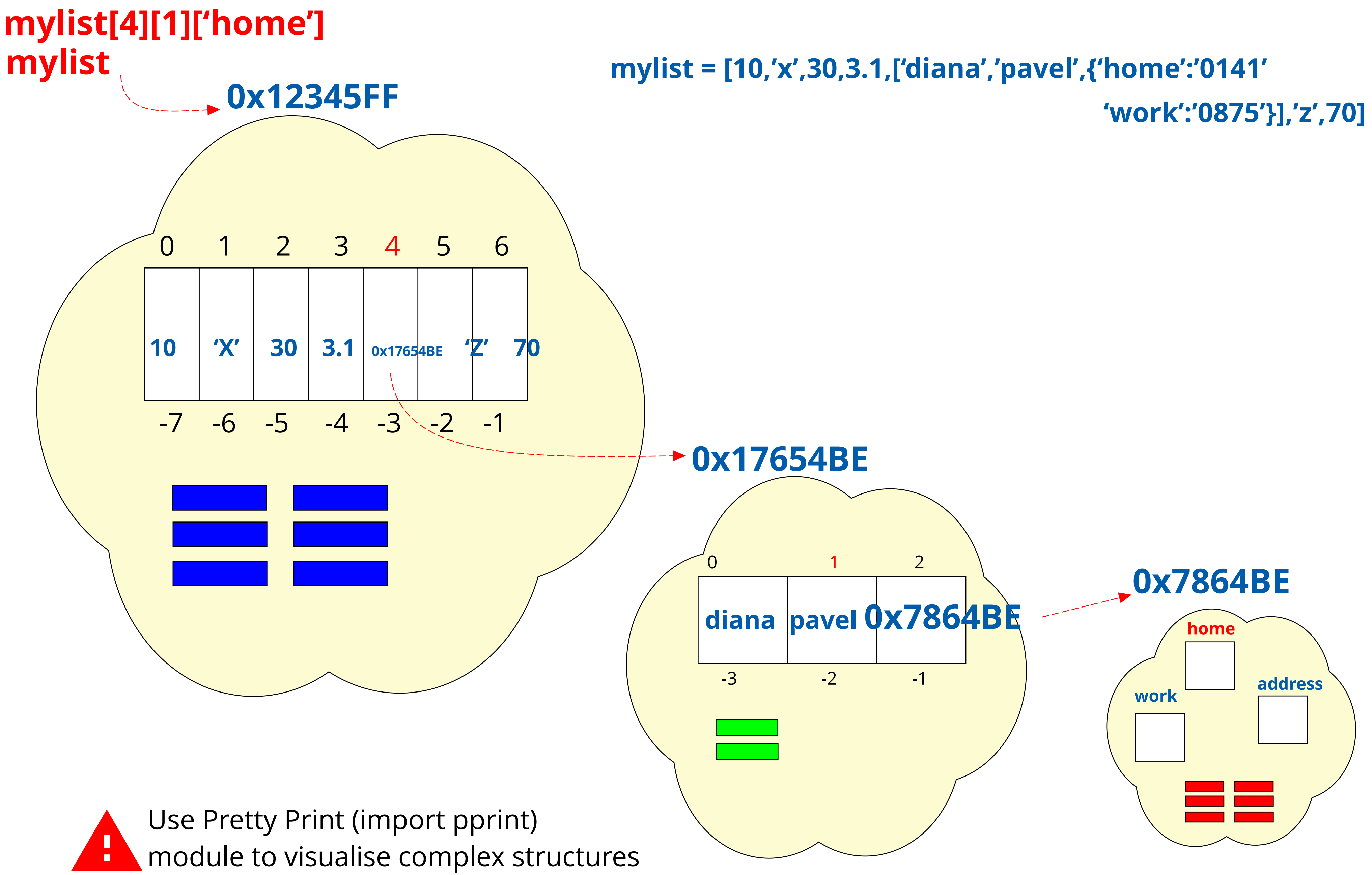


Ordered
Indexed using [n]
Raw Binary Data(2.6)
Immutable

```
mybytes = bytes('hello', 'utf-8')
mybytes = b'hello'
mybytes.isalpha()
mybytes.decode()
mybytes[-1]
```

Ch5: Collections - Multi-Dimensional Structures

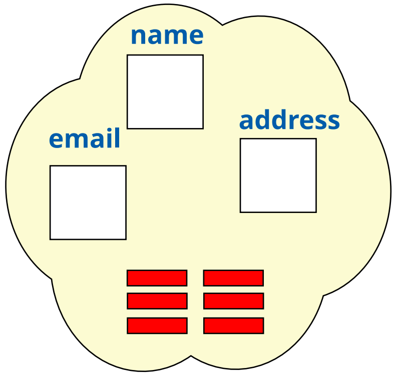
mylist = [10,'x',30,3.1,['diana','pavel',{'home':'0141' 'work':'0875'}],'z',70]



Use Pretty Print (import pprint) module to visualise complex structures

Ch5: Collections - Un-ordered Sequences

dict



Unordered

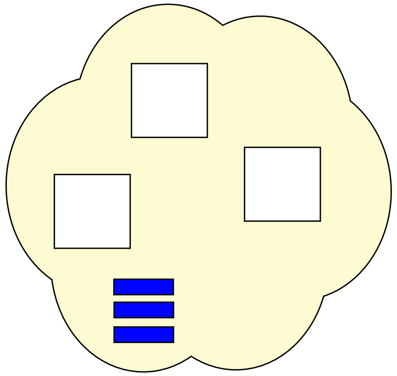
Indexed using ['key']

Mutable

Dynamic, fast searching

Keys must be unique

set

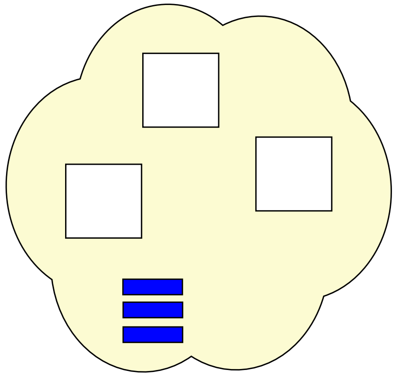


Unordered

Unique data values

Mutable

frozenset



Unordered

Unique data values

Immutable

Ch6: Regular Expressions

regex101.com - a good place to practice!

```
UNIX: $ grep ‘pattern$’ file1 file2 *
```

B.R.E/Regex/E.R.E

Line Anchors

start	‘^the’
end	‘ing$’

Single Char Class

‘.’ = match 1 any char

‘.onald’

‘^..........$’ = match 10char lines

Limited range char class

‘[abc]’ = match 1 x list chars

‘[a-z]’ = match 1 x range chars

‘[a-zA-Z]’ = match 1 x range char

‘[^0-9]’ = match 1 x not range

‘[aeiou][aeiou][aeiou]’

‘[dD]onald’

‘[aeiou]{3}’

‘^.{10}$’

:+-15digits:

‘:[+-]?\d{15}:’

Escape Char

‘\.’ = escape next char

Repetition Char

‘e*’ = repeat 0 or more

_
e
ee
eee

‘[0-9]*’

‘[0-9][0-9]*’

‘.*’

‘..*’

‘^.*$’

‘e?’ = repeat 0 or once

‘e+’ = repeat 1 or more

‘(rhubarb)+’

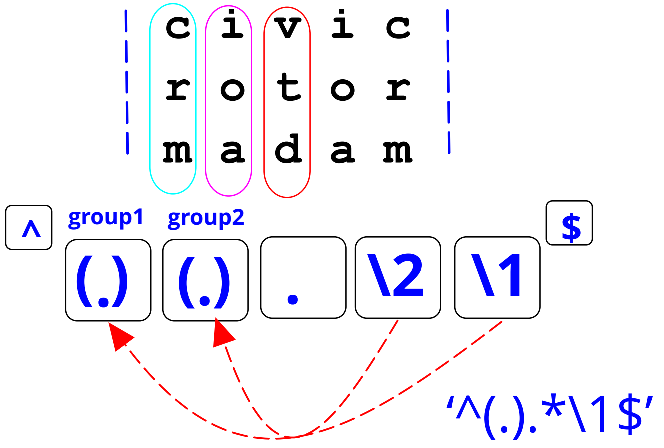
Quantifiers/Interval Repetitions

‘[0-9]{10}’	= Exact reps
‘[0-9]{10,20}’	= min,max reps
‘[0-9]{15,}’	= at least reps
‘[0-9]{0,15}’	= at most reps

Alternation (or)

‘donaldc|dcameron|Sir Donald’

Groupings/Back Referrals

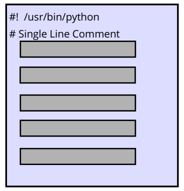


Python Escape/Shortcuts

\d - [0-9]	\D - [^0-9]
\w - [a-zA-Z0-9_]	\W - [^a-zA-Z0-9_]
\s - [\t\n\r\f]	\S - [^ \t\n\r\f]
\b - [\t:;,.?!’”<>]	\B - [^ \t:;,.?!’”<>]
\A - start of string	\Z - end of string

Ch6: Regular Expression Match

Python Standard library

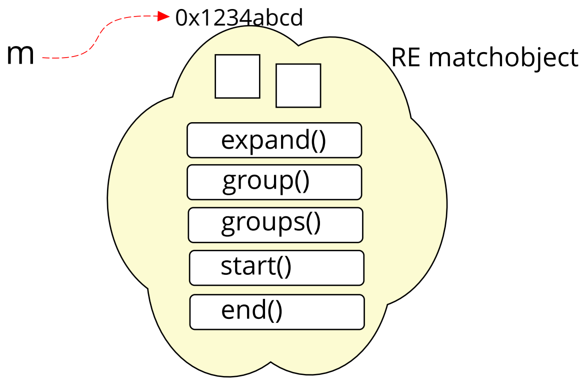


re.py

```
#! /usr/bin/python
# Single Line Comment

import  re

argc = len(sys.argv)
if argc > 1:
   print(‘Too many args’)
else:
   where = ‘World’
   print(“Hello”, where)

print(‘Goodbye from  ‘ +
sys.argv[0])
```

```
line = “root:x:0:0:The Super User:/root:/bin/bash”

m = re.search(r”^(root).*\1”, line)
m = re.match(r”(root).*\1”, line)
m = re.fullmatch(r”^root.*\n$”, line)
type(m)
```

} returns matchObject or None



```
if m:
   print(m.start(), m.end() )
else:
   print(“No match”)
```

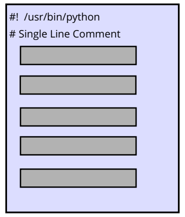
print(m.group())	returns matched string
print(m.groups())	returns tuple of matched group members
print(m.groups()[0])	returns match from group 1 (yuck!)
print(m.group(1))	returns match from group 1 (Pythonic)

consider pre-compiling pattern

```
reobj = re.compile(r”^(root).*\1”)
m = reobj.search(line)
```

```
reobj = re.compile(r”^(root).*\1”)
for somethings in biglist:
   m= reobj.search(something)
```

Ch6: Regular Expression Substitution



re.py

```
#!  /usr/bin/python
# Single Line Comment

import  re

argc = len(sys.argv)
if argc > 1:
   print(‘Too many args’)
else:
   where = ‘World’
   print(“Hello”, where)

print(‘Goodbye from  ‘ +
sys.argv[0])
```

line = “root:x:0:0:The Super User:/root:/bin/bash”

line = re.sub(r”[Ss]uper [uU]ser”, r”Administrator”, line) returns modified string

(line, num) = re.subn(r”[Ss]uper [uU]ser”, r”Administrator”, line)

returns tuple (string, num changes)

fields = re.split(r‘[:;.,]’, line) returns list

print(fields)

Flags - change behaviour of match

Long name	Short	RE	
`re.IGNORECASE`	`re.I`	`(?i)`	Case insensitive match
`re.MULTILINE`	`re.M`	`(?m)`	^ and $ match start and end of *line*
`re.DOTALL`	`re.S`	`(?s)`	. also matches a new-line
`re.VERBOSE`	`re.X`	`(?x)`	Whitespace is ignored, allow comments

m = re.search(r”(?im)^(root).*\1”, line) modifier applies to entire pattern

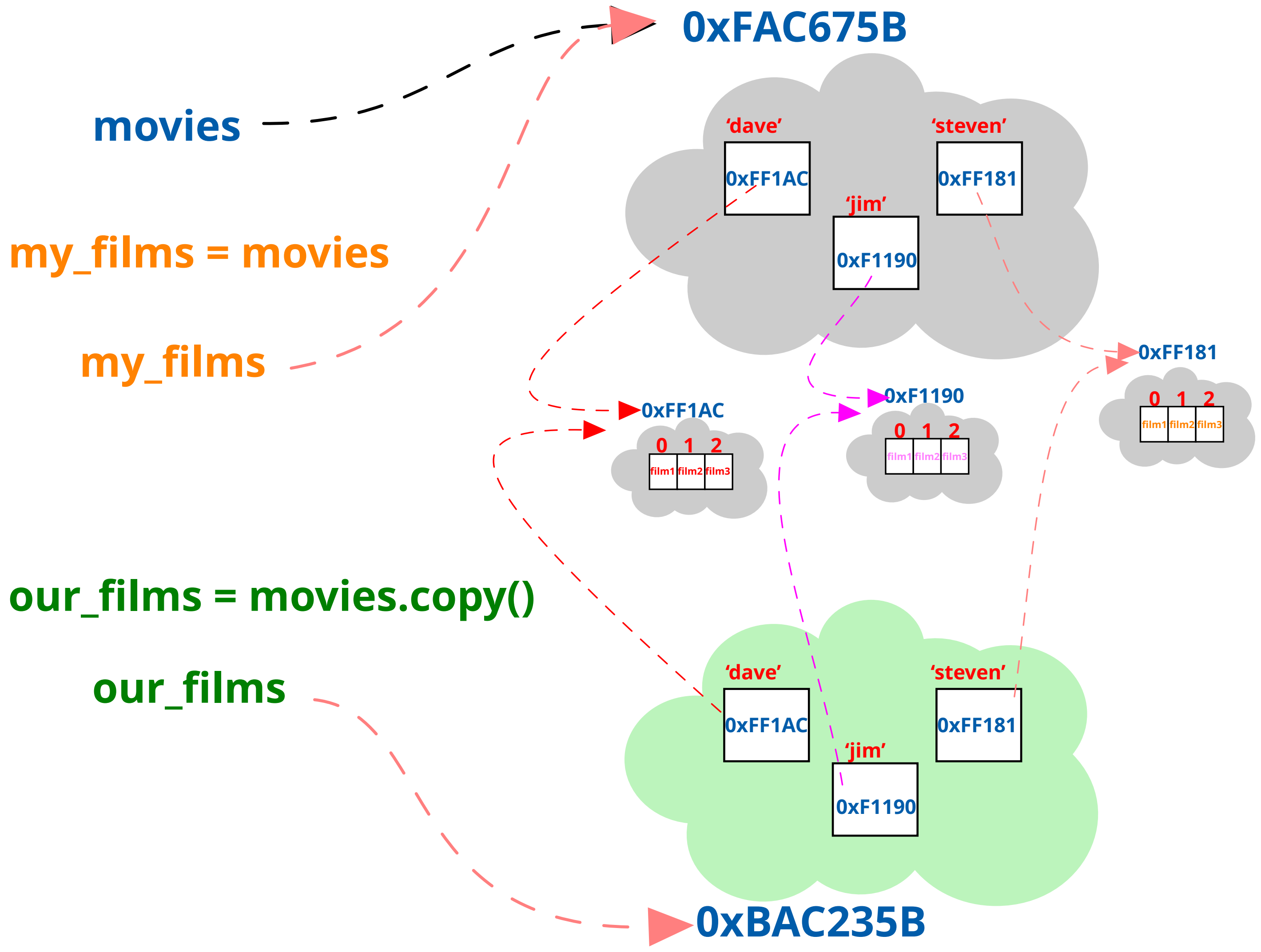
m = re.match(r”(root).*\1”, line, flags=re.IGNORECASE|re.MULTILINE)

m = re.fullmatch(r”^root.*$”, line)

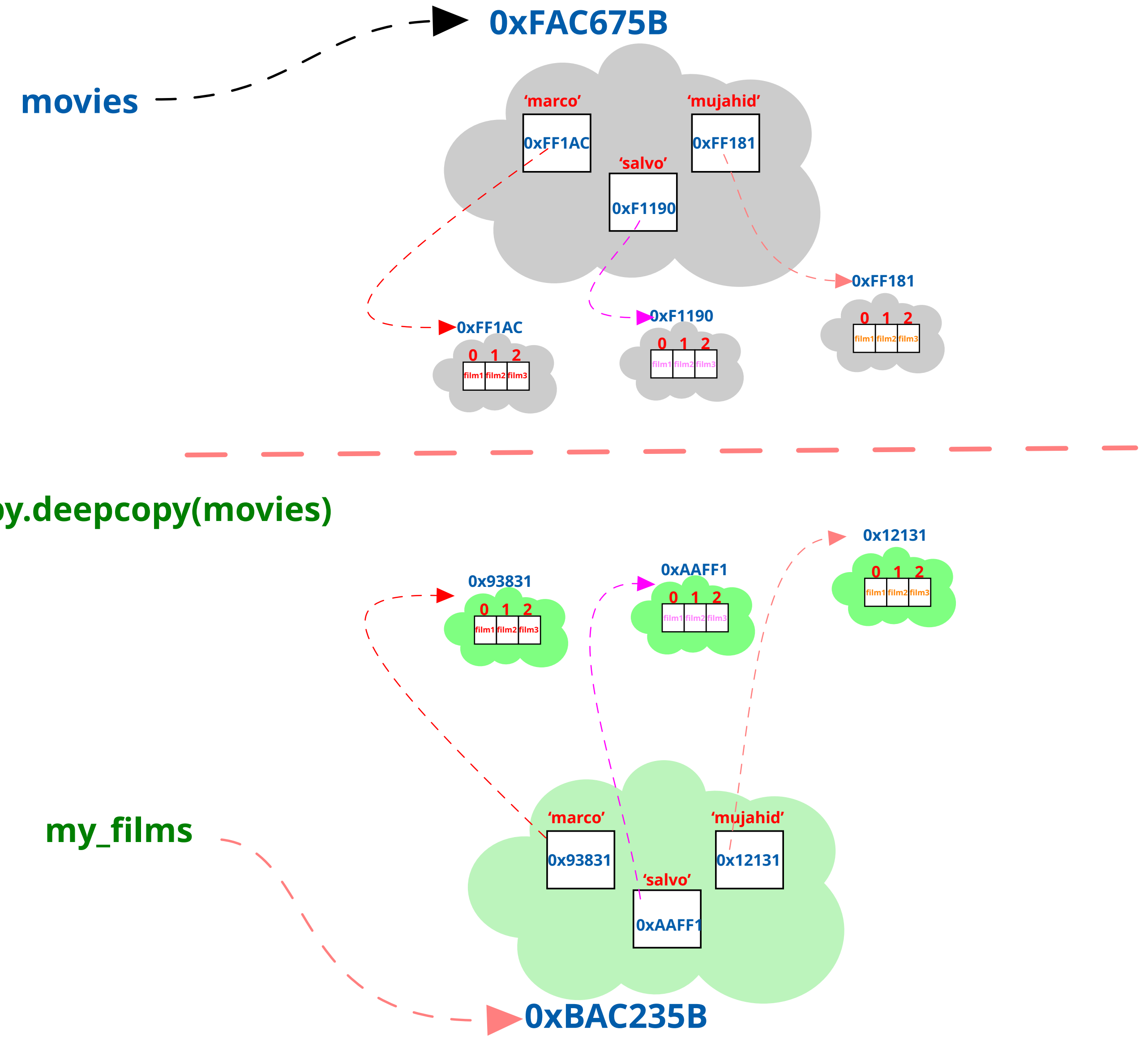
m = re.sub(r”(?i:s)uper (?i:u)ser”, ”Administrator”, list)

modifier span (Py3.6)

Ch9: Advanced Collection - Copying references and shallow copy



Ch8: Advanced Collection - Deep Copy

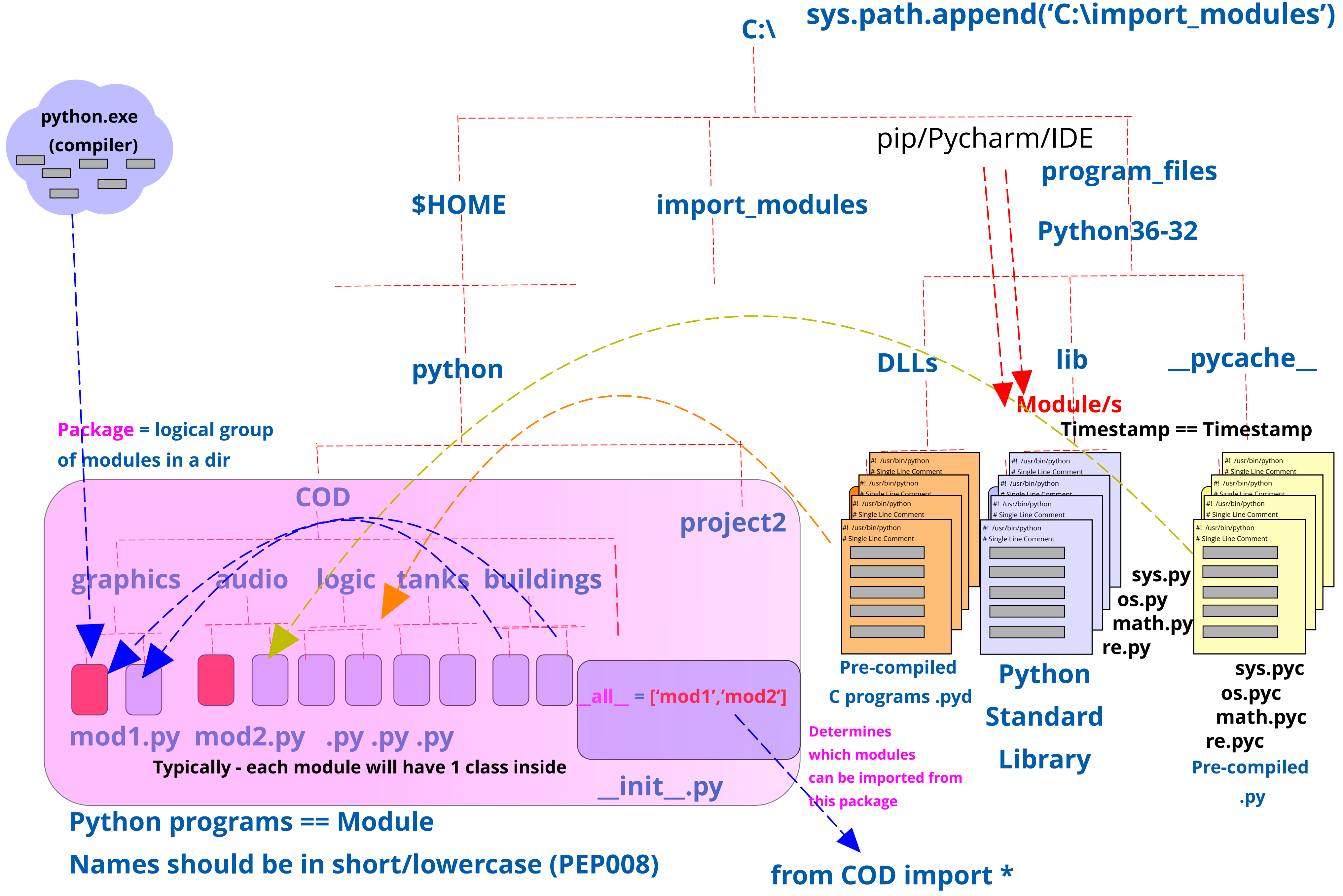


```
import copy
my_films = copy.deepcopy(movies)
```

Ch10: Modules and Packages

$PYTHONPATH

```
import sys
sys.path.append('C:\import_modules')
```



Python programs == Module

Names should be in short/lowercase (PEP008)

Ch10: Namespaces

$PYTHONPATH

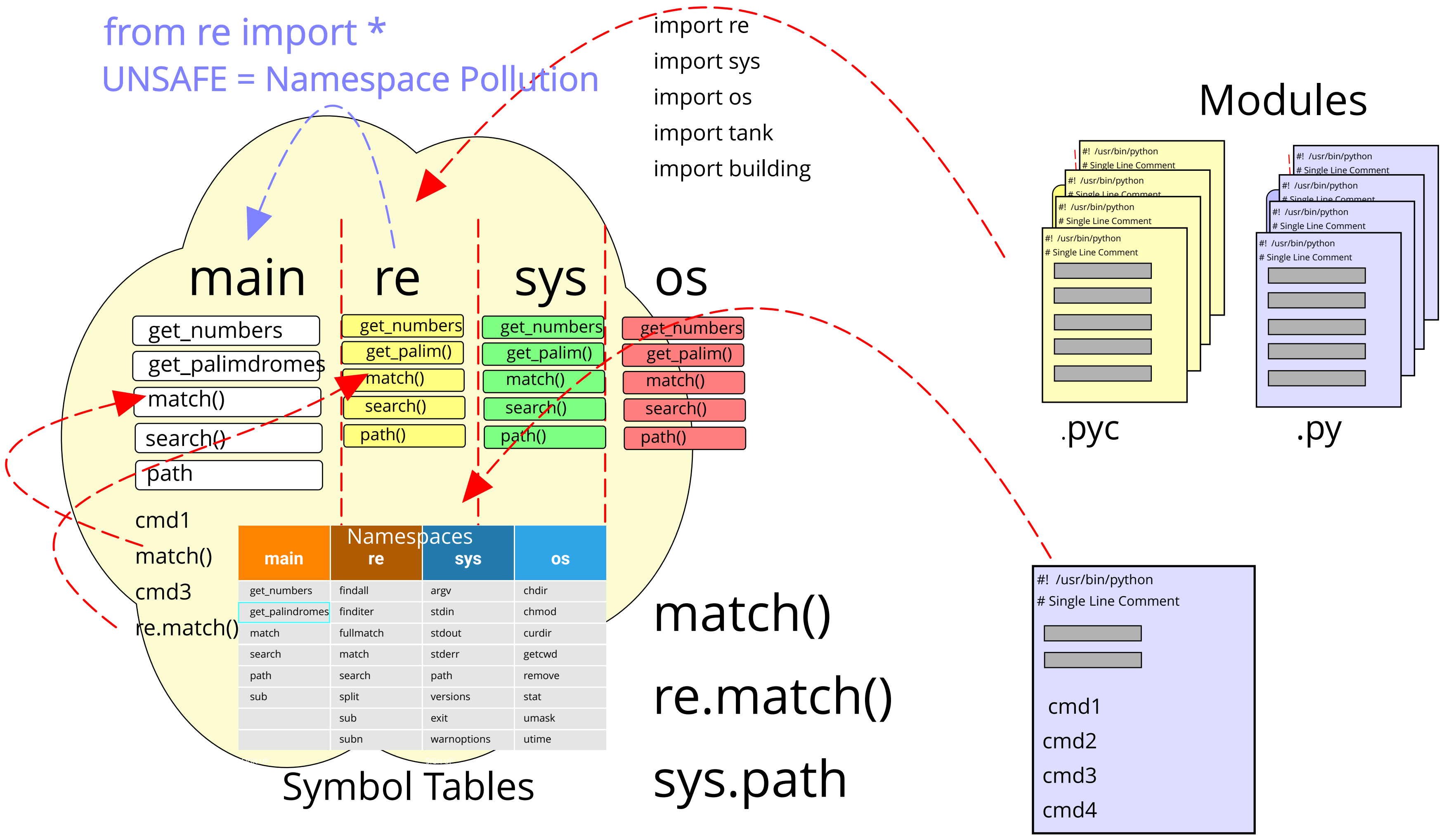
import sys

sys.path.append('C:\import_modules')

from re import search, match

from re import *

UNSAFE = Namespace Pollution



Namespaces

main	re	sys	os
get_numbers	findall	argv	chdir
get_palindromes	finditer	stdin	chmod
match	fullmatch	stdout	curdir
search	match	stderr	getcwd
path	search	path	remove
sub	split	versions	stat
	sub	exit	umask
	subn	warnoptions	utime

Symbol Tables

match()

re.match()

sys.path

cm5

Ch10: Namespaces

$PYTHONPATH

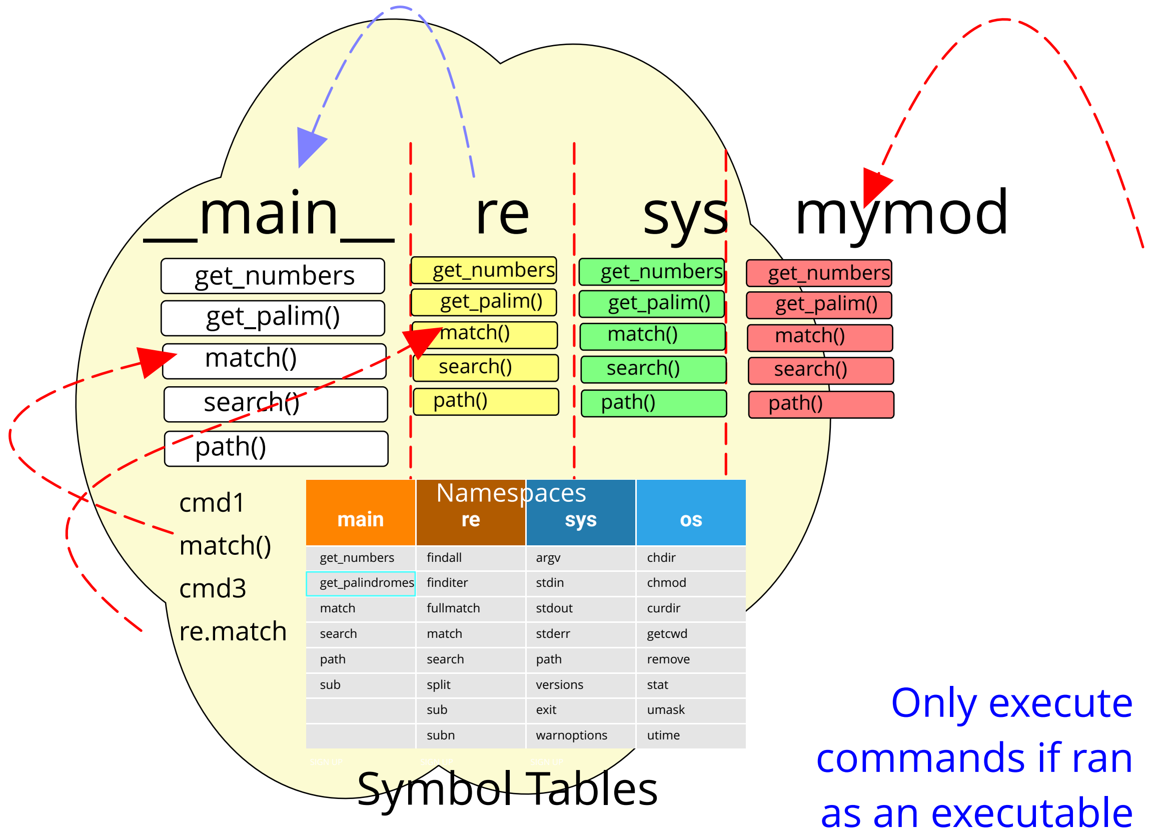
```
import sys
sys.path.append('C:\import_modules')
```

from re import search, match

from re import *

UNSAFE = Namespace Pollution

import mymod



main	re	sys	os
get_numbers	findall	argv	chdir
get_palindromes	finditer	stdin	chmod
match	fullmatch	stdout	curdir
search	match	stderr	getcwd
path	search	path	remove
sub	split	versions	stat
	sub	exit	umask
	subn	warnoptions	utime

Namespaces

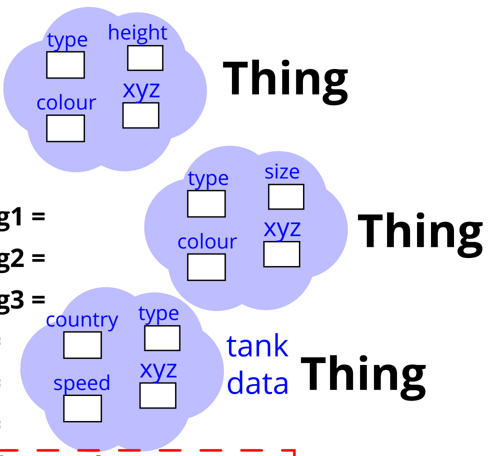
Symbol Tables

mymod

```
#!  /usr/bin/python
# Single Line Comment

def func1():
    ------
def func2():
    ------
def main():
    cmd1
    cmd2
    cmd3

if __name__  == "__main__":
    main()
```

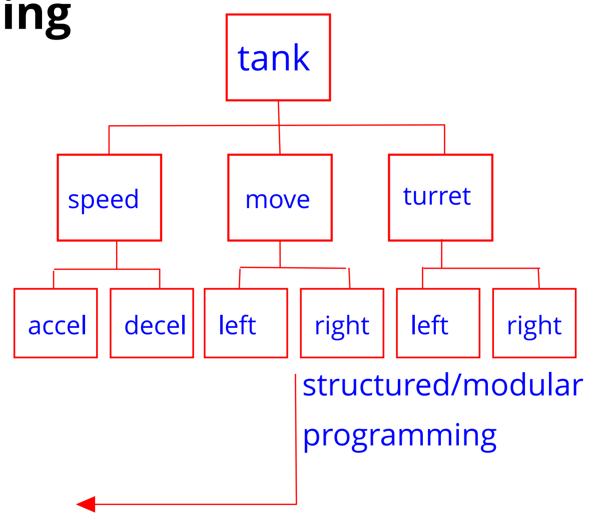
Only execute commands if ran as an executable

Ch11: Traditional Functional/Procedural Programming



```
tree1 =
tree2 =
tree3 =
tree4 =
building1 =
building2 =
building3 =
tank1 =
tank2 =
tank3 =
def tank1_accel() :
        return
def tank1_decel() :
        return
def tank1_rotate_left() :
        return
def tank1_rotate_right() :
        return
def tank1_rotate_turret_left() :
        return
def tank1_shoot() :
        return
def tank1_health_status():
        return
procedure main_game() {
        join_game()
        create_tank1()
        while still_alive:
                game_logic()
```

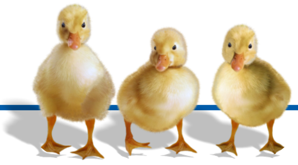
tank
behaviour

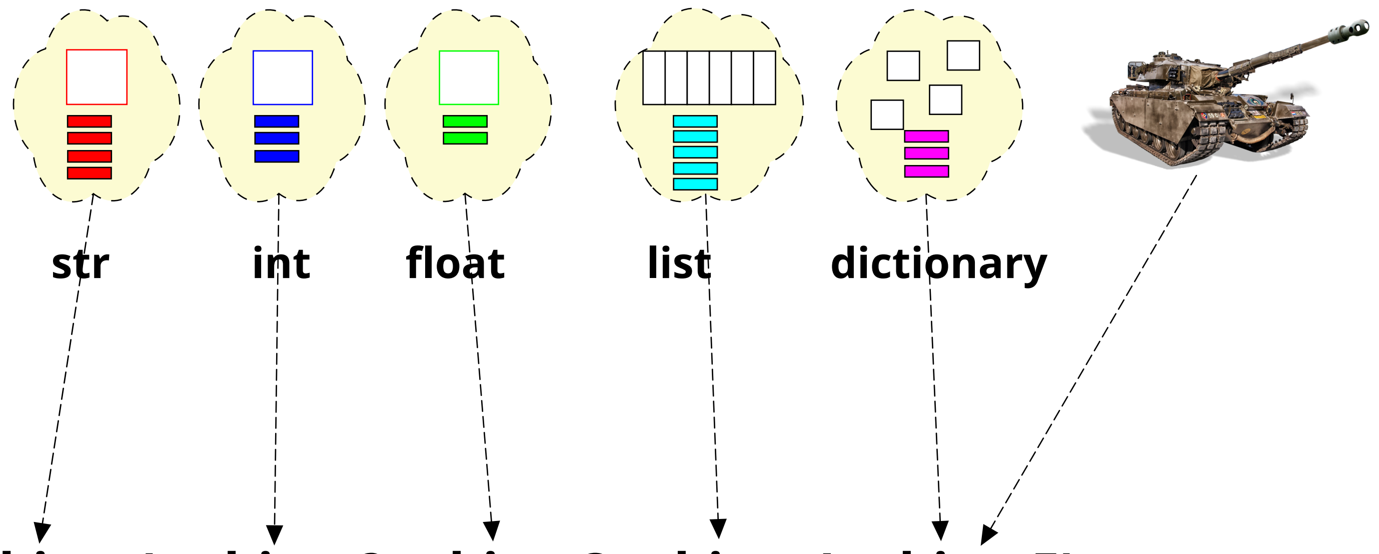


structured/modular programming

replicate for every tree, building, rock, soldier and tank...

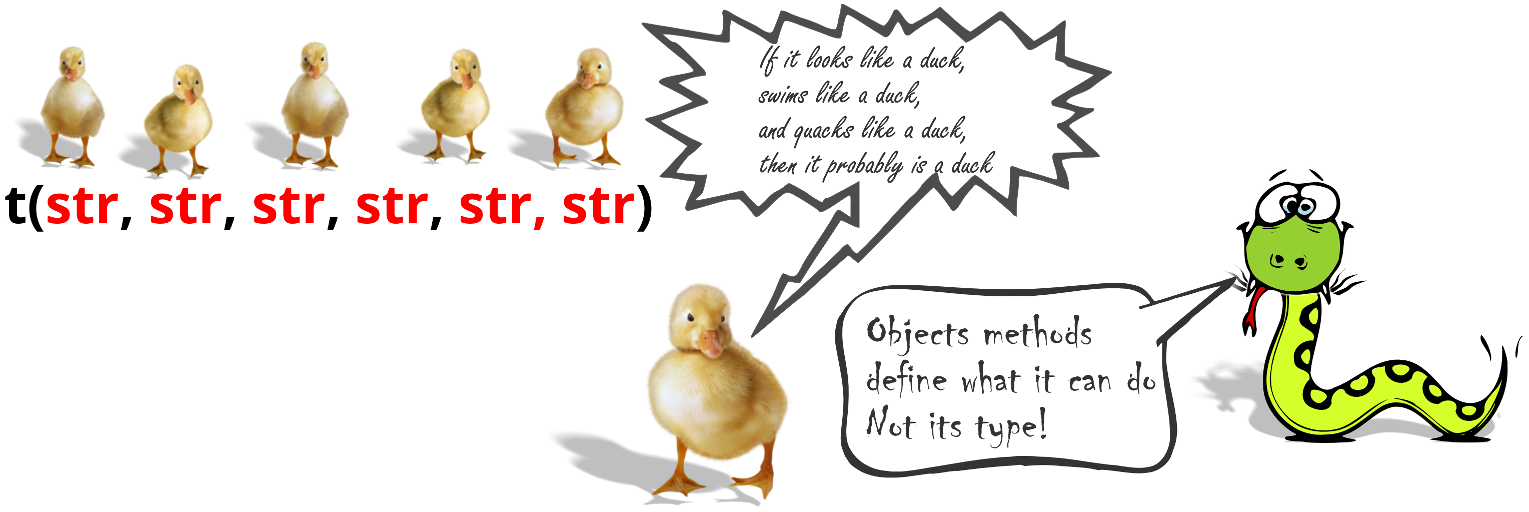
...results in overly large, complex source code file





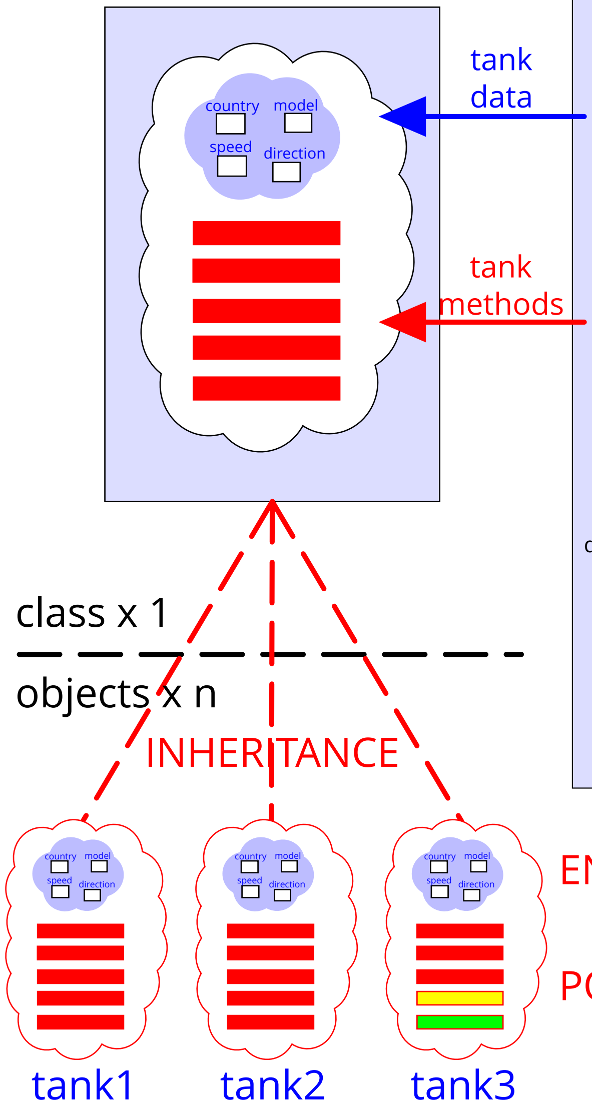
print(object1, object2, object3, object4, object5)

print(str, int, float, list, dictionary, Tank)



print(str, str, str, str, str, str)

Ch11: Object Oriented Programming



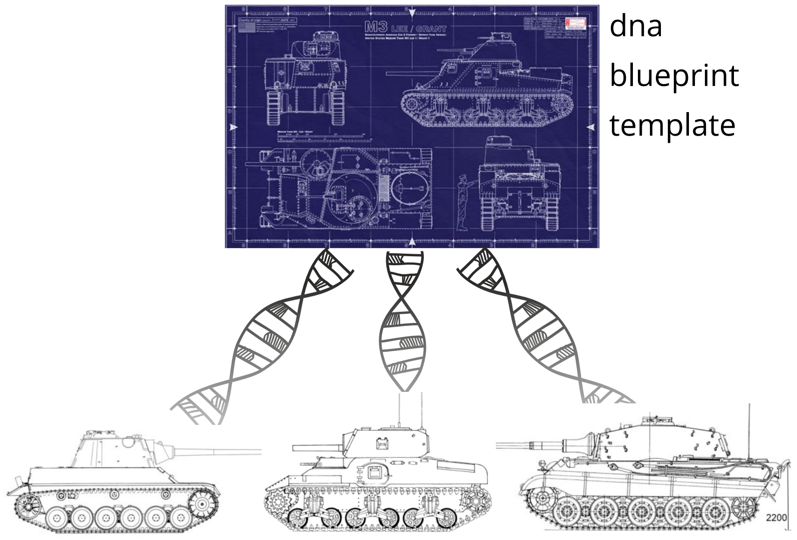
```
#! /usr/bin/python

class Tank():
    def __init__(self, country, model):
        self.attr['country'] = ""
        self.attr['model'] = ""
        self.attr['speed'] = 0
    def accelerate(self, amount):
        self.attr['speed'] += amount
        return
    def decelerate(self, amount):
        self.attr['speed'] -= amount
        return
    def rotate_left(self, degrees):
        self.attr['direction'] -= degrees % 360
        return

def main():
    join_game()
    tank1 = Tank('german', 'tiger')
    tank2 = Tank(american, sherman')
    tank3 = Tank('british', 'churchill')
    while still_alive:
        game_logic()
```

ENCAPSULATION

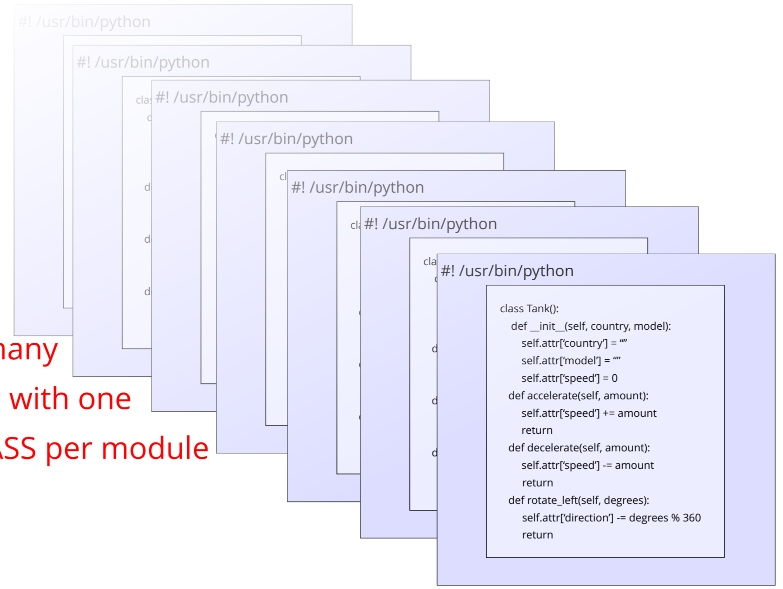
POLYMORPHISM

Created @ runtime

class Tank



"Real" objects with Attributes and Behaviour


#! /usr/bin/python

class Tank():
    def __init__(self, country, model):
        self.attr['country'] = ""
        self.attr['model'] = ""
        self.attr['speed'] = 0
    def accelerate(self, amount):
        self.attr['speed'] += amount
        return
    def decelerate(self, amount):
        self.attr['speed'] -= amount
        return
    def rotate_left(self, degrees):
        self.attr['direction'] -= degrees % 360
        return

Often have many modules with one CLASS per module

...results in smaller, less complex and more resuable code

Ch12: Error Handling and Exceptions

```
#! /usr/bin/python
# comments
"""
        This is a docstring describing what
        the program/module/class does
"""
import sys

try:
    filename = r"C:\labs\words.txt"
    fh_in = open(filename, "r")
except FileNotFoundError as err:
    print(f"Error whilst opening file {err.filename}", file=sys.stderr)
    print(f"Error: {err.args[1]} Code: {err.args[0]}",  file=sys.stderr)
except PermissionError as err:
    print(f"Error Permission denied on {err.filename}", file=sys.stderr)
    print(f"Error: {err.args[1]} Code: {err.args[0]}",  file=sys.stderr)
else:
    print(f"Successfuly opened file {filename}")
```

if something could fail,
wrap it in a try/except block

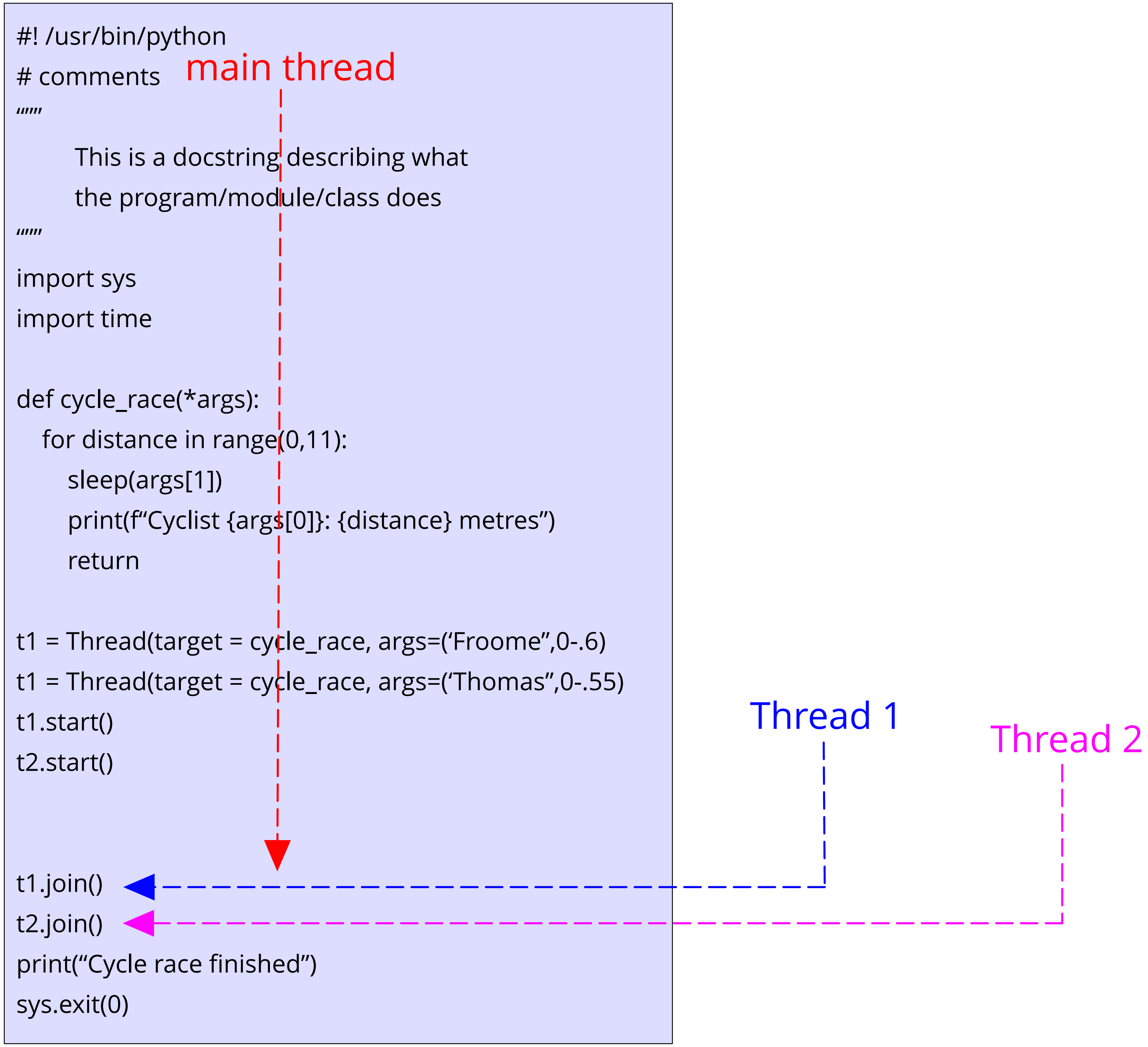
```
BaseException
+-- SystemExit
+-- KeyboardInterrupt
+-- GeneratorExit
+-- Exception
    +-- StopIteration
    +-- StopAsyncIteration
    +-- ArithmeticErr
        +-- FloatingP
        +-- OverflowE
        +-- ZeroDivis
    +-- AssertionErro
    +-- AttributeErro
    +-- BufferError
    +-- EOFError
    +-- ImportError
        +-- ModuleNot
    +-- LookupError
        +-- IndexErro
        +-- KeyError
    +-- MemoryError
    +-- NameError
        +-- UnboundLo
```

```
+-- Exception
    ....
    +-- OSError
        +-- BlockingIOError
        +-- ChildProcessError
        +-- ConnectionError
            +-- BrokenPipeError
            +-- ConnectionAbortedError
            +-- ConnectionRefusedError
            +-- Conne
        +-- FileExists
        +-- FileNotFou
        +-- Interrupte
        +-- IsADirecto
        +-- NotADirect
        +-- Permission
        +-- ProcessLoc
        +-- TimeoutErr
```

```
+-- Exception
....
    +-- ReferenceError
    +-- RuntimeError
        +-- NotImplementedError
        +-- RecursionError
    +-- SyntaxError
        +-- IndentationError
            +-- TabError
    +-- SystemError
    +-- TypeError
    +-- ValueError
        +-- UnicodeError
            +-- UnicodeDecodeError
            +-- UnicodeEncodeError
            +-- UnicodeTranslateError
```

Exceptions are classes,
and have a hierarchy

Ch13: Multitasking - threads



```
#! /usr/bin/python
# comments
“””
        This is a docstring describing what
        the program/module/class does
“””
import sys
import time

def cycle_race(*args):
    for distance in range(0,11):
        sleep(args[1])
        print(f“Cyclist {args[0]}: {distance} metres”)
        return

t1 = Thread(target = cycle_race, args=(‘Froome”,0-.6)
t1 = Thread(target = cycle_race, args=(‘Thomas”,0-.55)
t1.start()
t2.start()


t1.join()
t2.join()
print(“Cycle race finished”)
sys.exit(0)
```