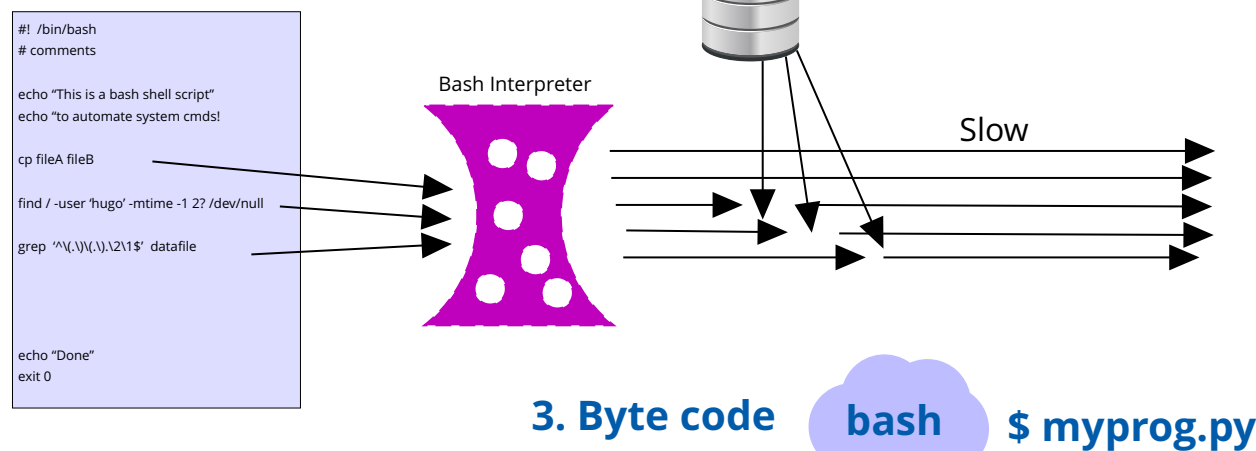
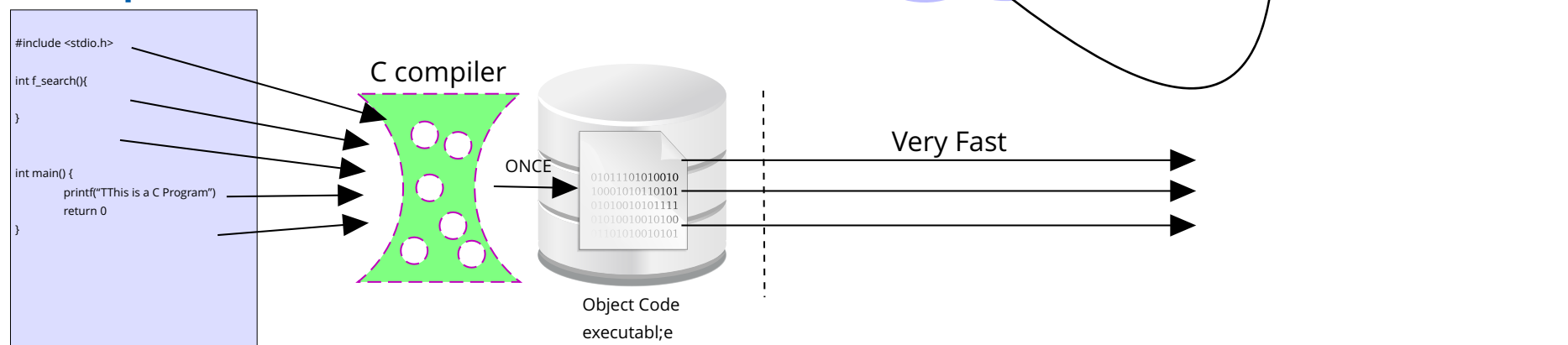
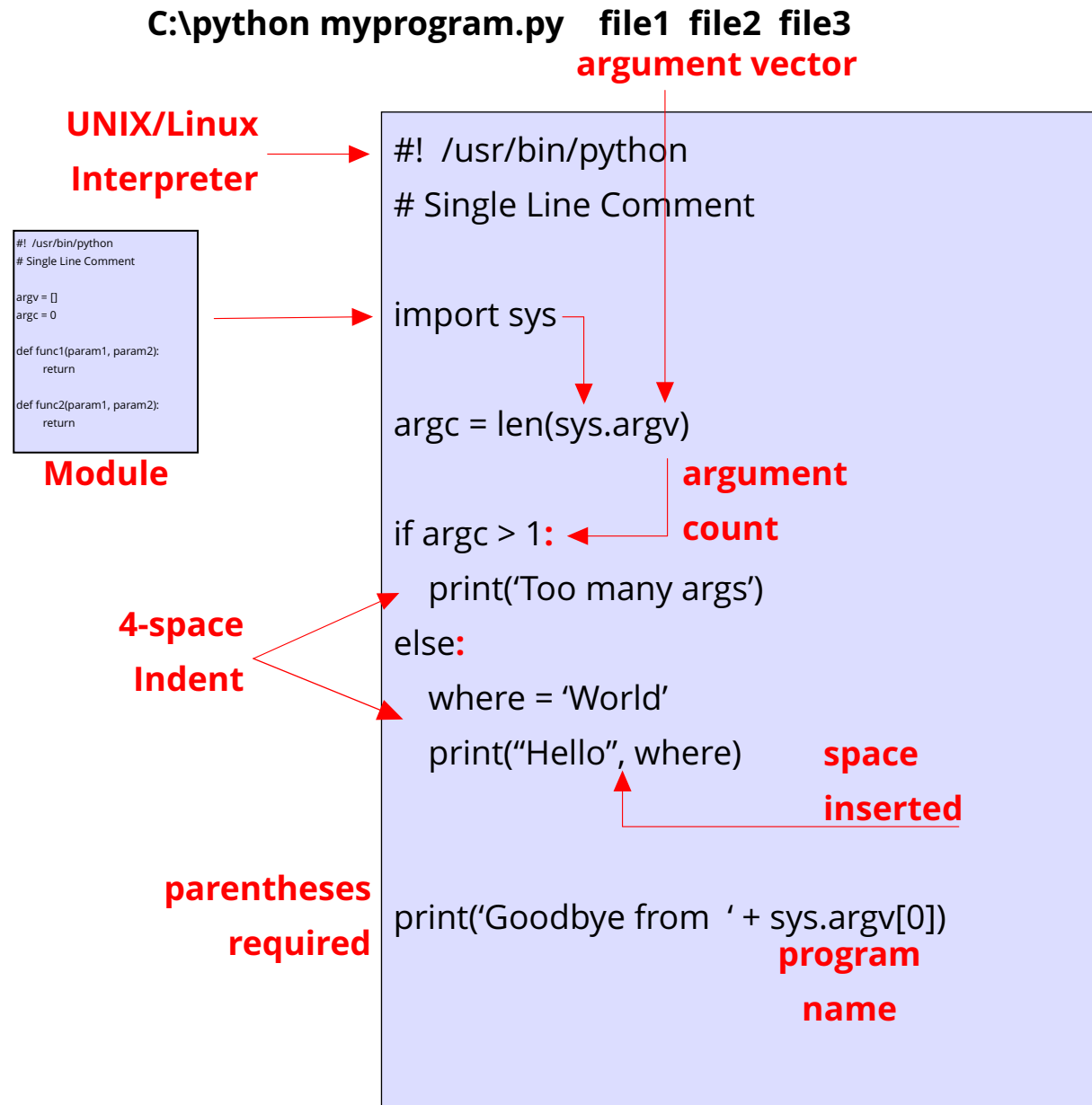


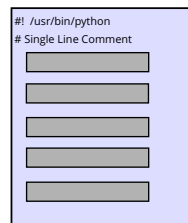
## 1. Interpreted



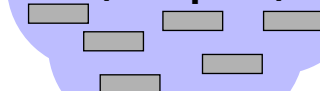
## 2. Compiled





**\_\_builtins\_\_****builtins.py**

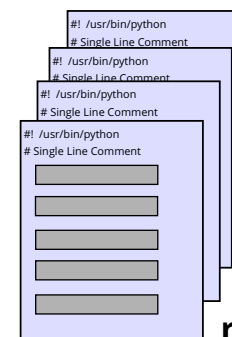
**abs(x)**  
**all(iterable)**  
**any(iterable)**  
**ascii(object)**  
**bin(x)**  
**bool([x])**  
**bytearray([arg[, encoding[, errors]]])**  
**bytes([arg[, encoding[, errors]]])**  
**callable(object)**  
**chr(i)**  
**classmethod(function)**  
**compile(source, filename, mode[, flags[, dont\_inherit]])**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

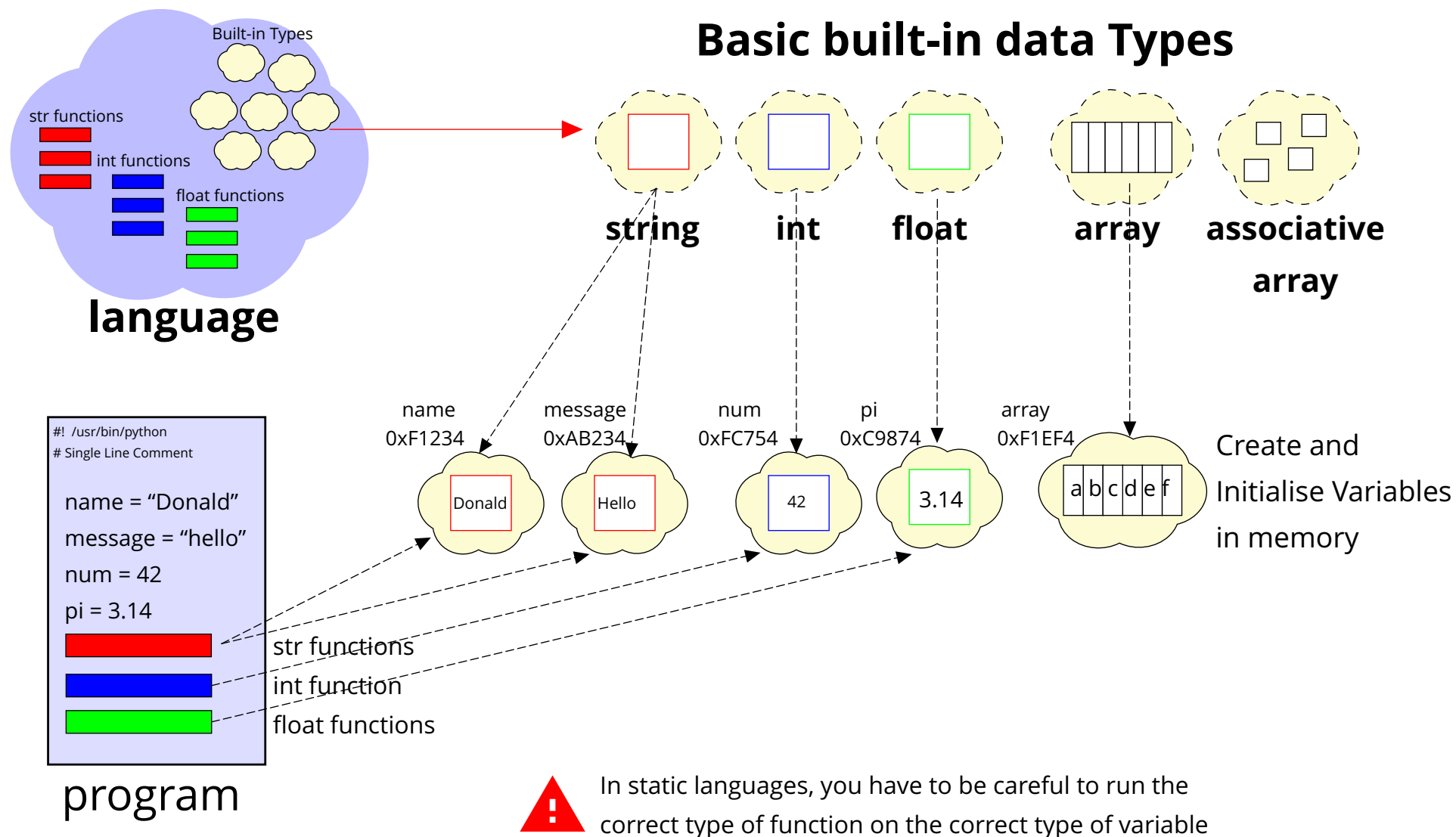
**python.exe**  
**(compiler)**

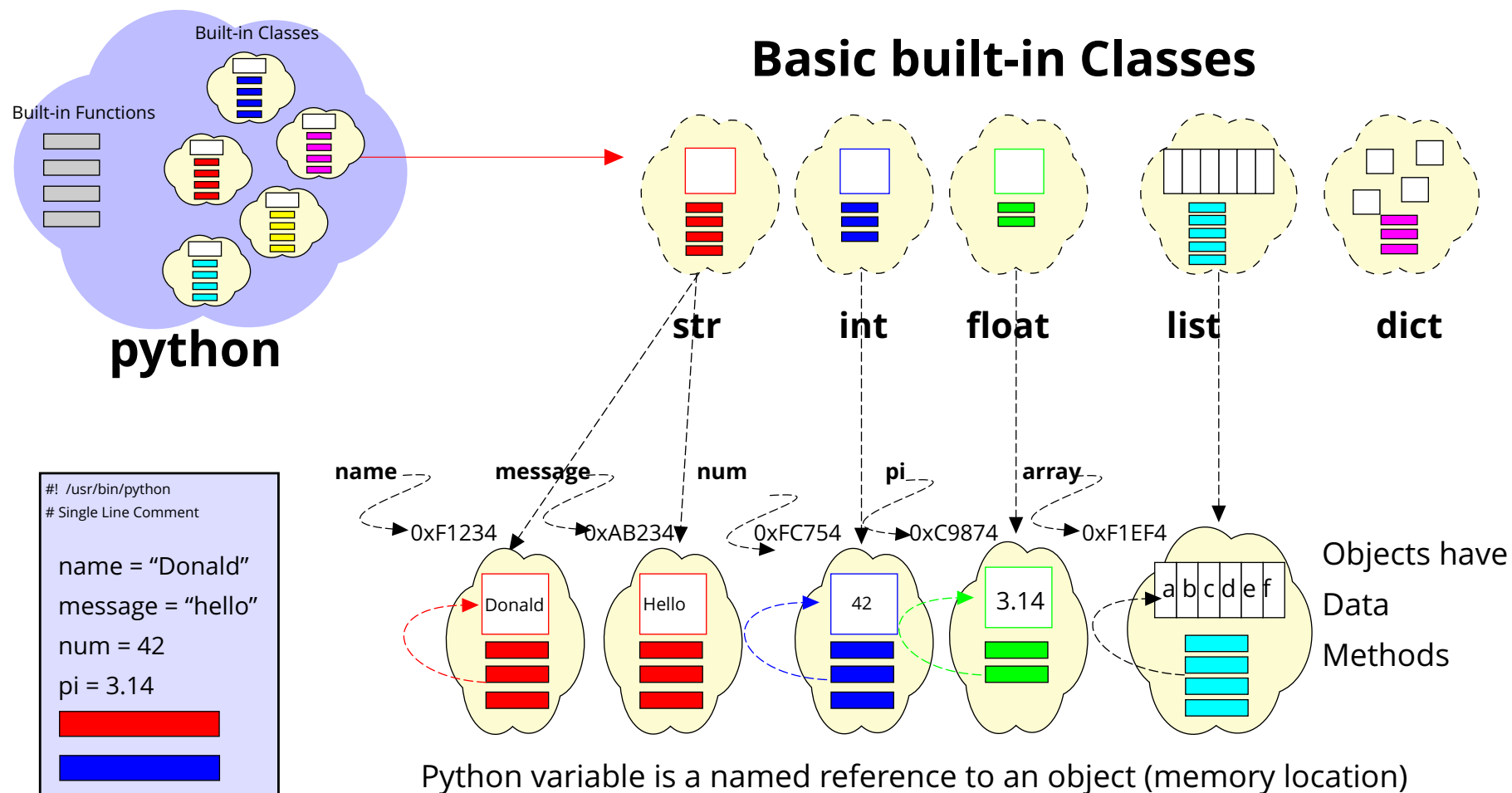
```
#!/usr/bin/python
# Single Line Comment
import sys, os
import math, re

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

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

**Python standard library**<https://docs.python.org/3/library/>**sys.py**  
**os.py**  
**math.py**  
**re.py****Module/s**





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

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

program

>>> help()

>>> help(str)

>>> help(int)

>>> help()

>>> help(str.lower)

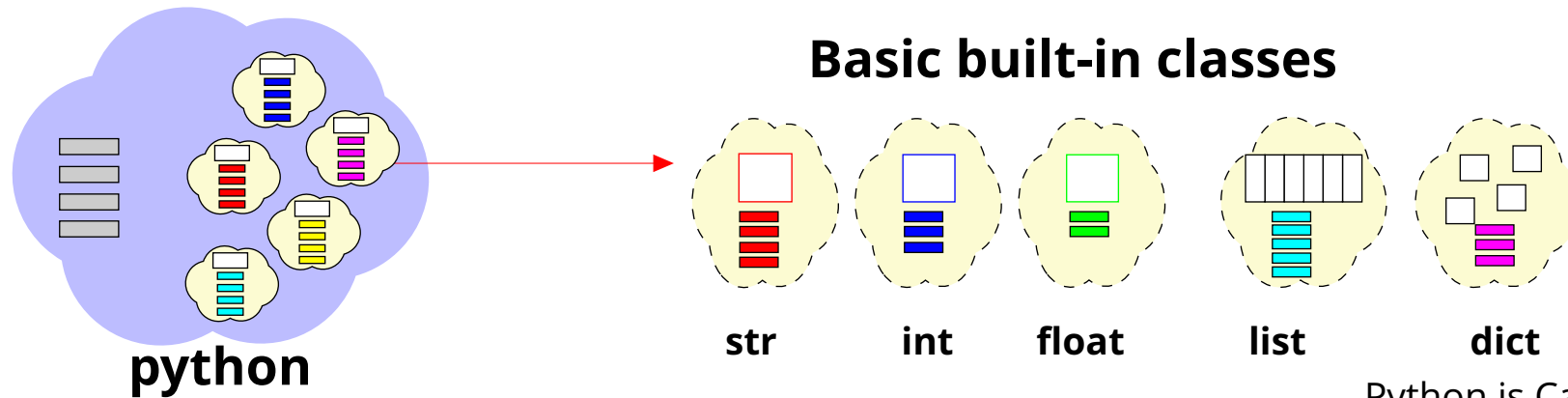
>>> type(name)

>>> dir()

>>> dir(str)

>>> dir(list)

>>> dir(name)



Python is CaSe sensitive

CASE CONVENTION (Style Guide)

taxrate	lowercase
TaxRate	ProperCase
taxRate	camelCase
tax_rate	snake_case

### Scope

<code>_myvar</code>	private(module/class)
<code>_myvar</code>	mangled(protected)
<code>__myvar__</code>	(special)

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

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

program

GLOBAL SCOPE



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

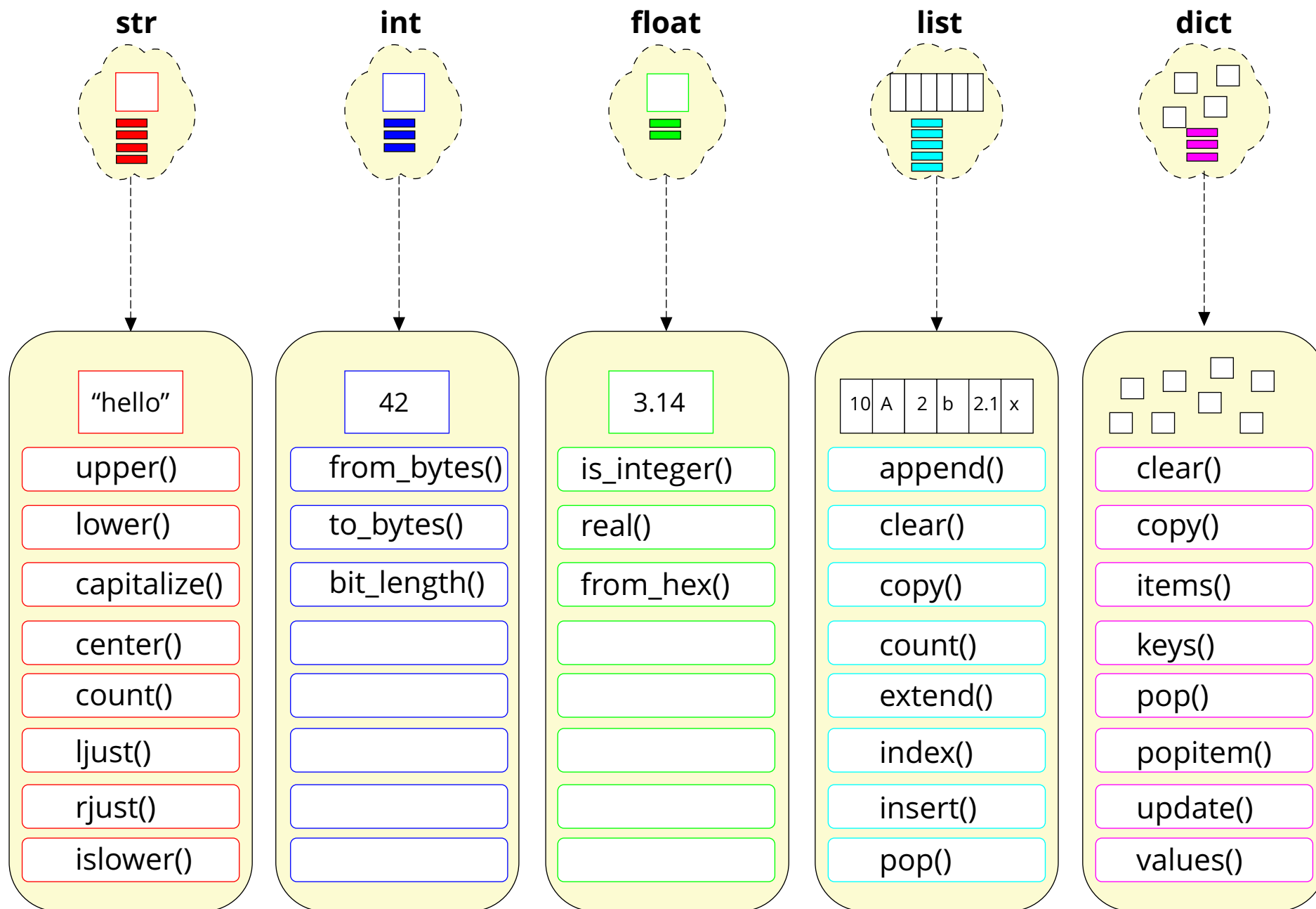
name = "Donald"
message = "hello"
num = 42
pi = 3.14
```

LOCAL SCOPE VARIABLE/OBJECT

`globals()`

`locals()`

1. Life
2. Visibility



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



Ordered sequence, mutable, dynamic in size

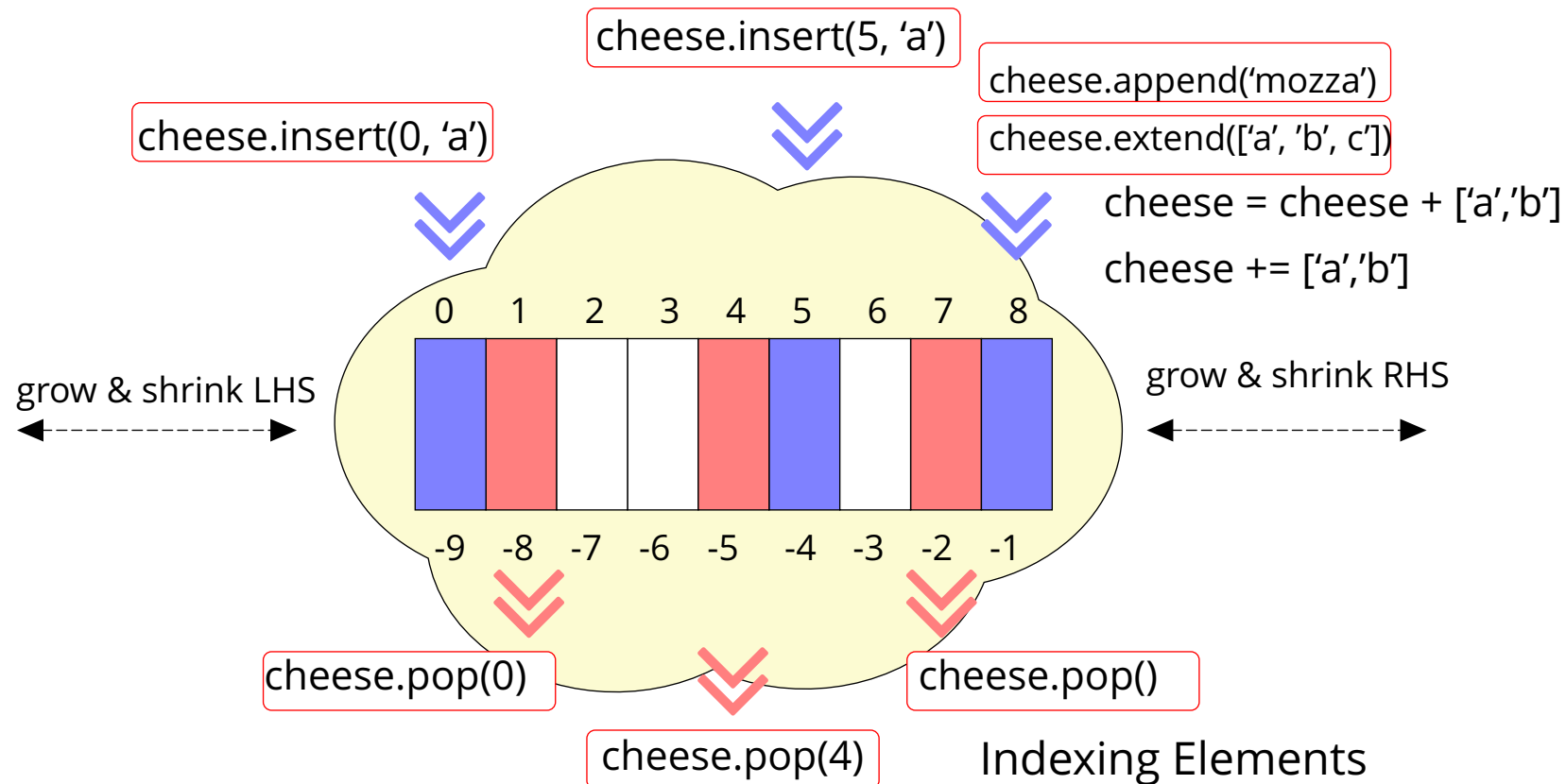
```
>>> cheese = ['cheddar', 'stilton', 'edam', 'gorgonzola']
```

```
>>> 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

Ordered sequence, mutable, dynamic in size

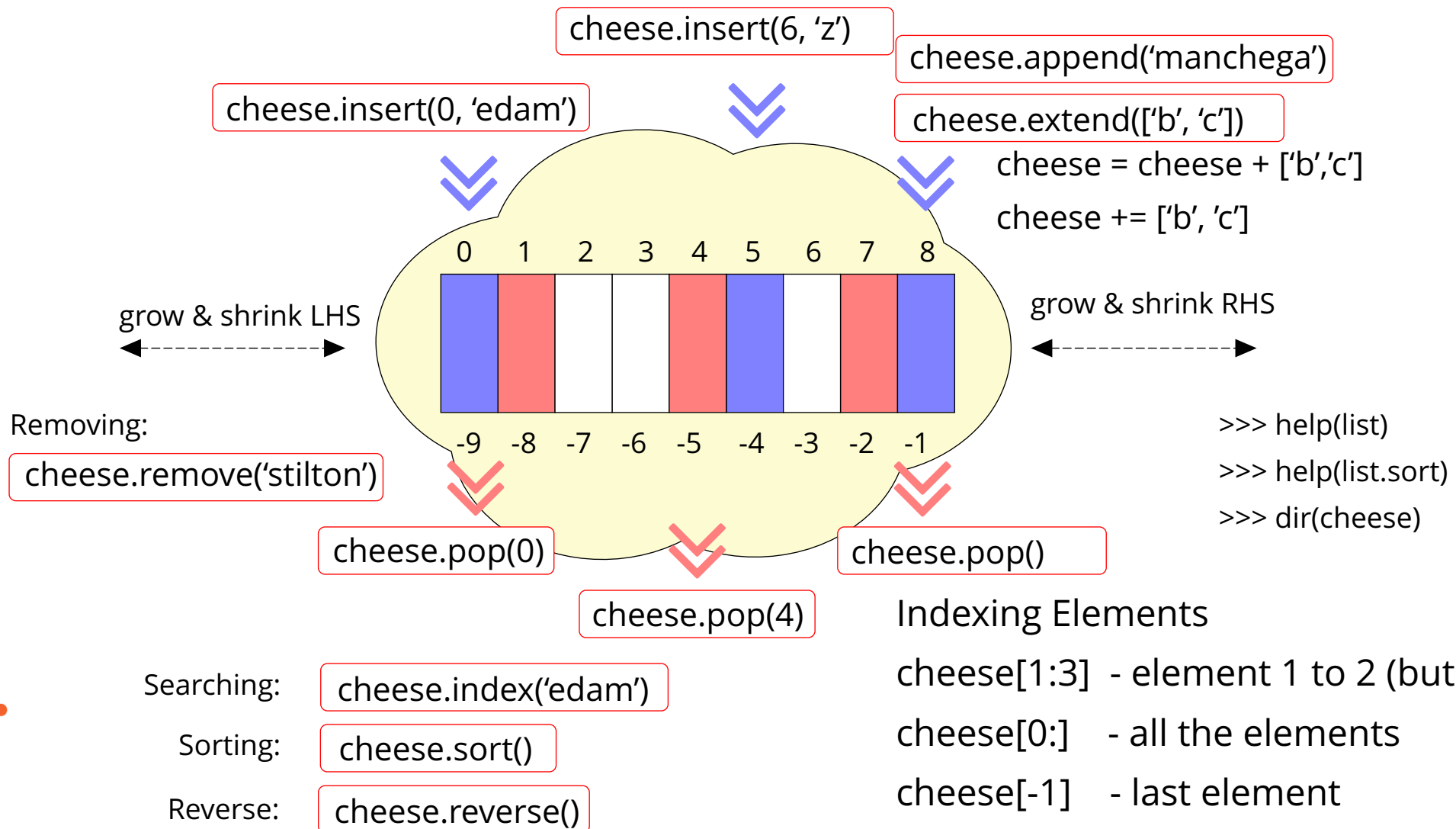
```
>>> cheese = ['cheddar', 'stilton', 'edam', 'gorgonzola']
```

```
>>> print(cheese[2])
```

'edam'

```
>>> print(cheese[-1])
```

'gorgonzola'

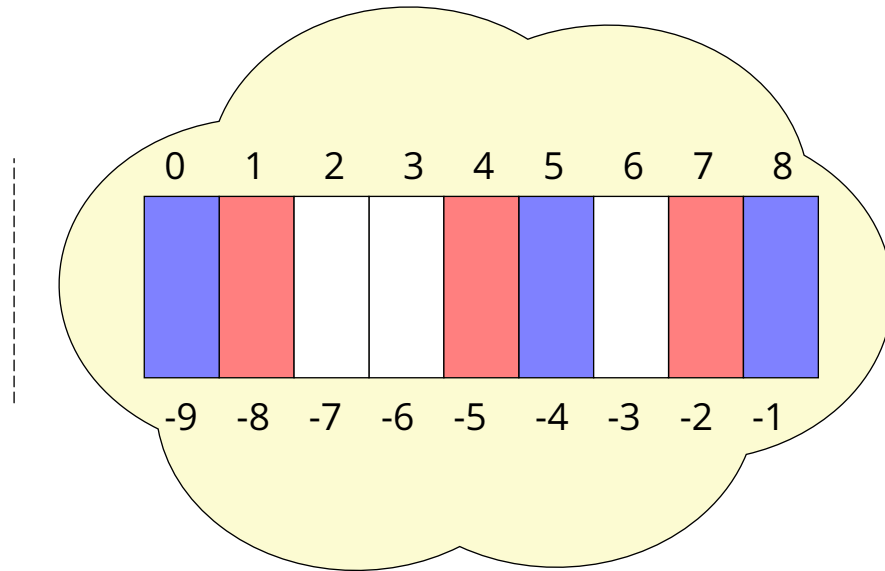


Ordered sequence, immutable

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

```
help(tuple)
```

```
dir(tuple)
```



```
>>> print(mytuple[1])
```

'bacon'

```
>>> print(mytuple[-1])
```

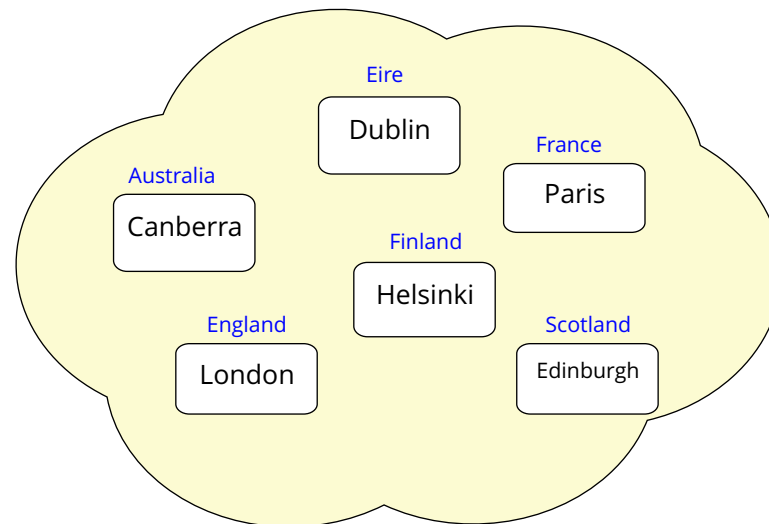
'tea'

```
>>> mytuple[2] = 'muffin'
```



Unordered sequence, mutable and dynamic in size, faster searching

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

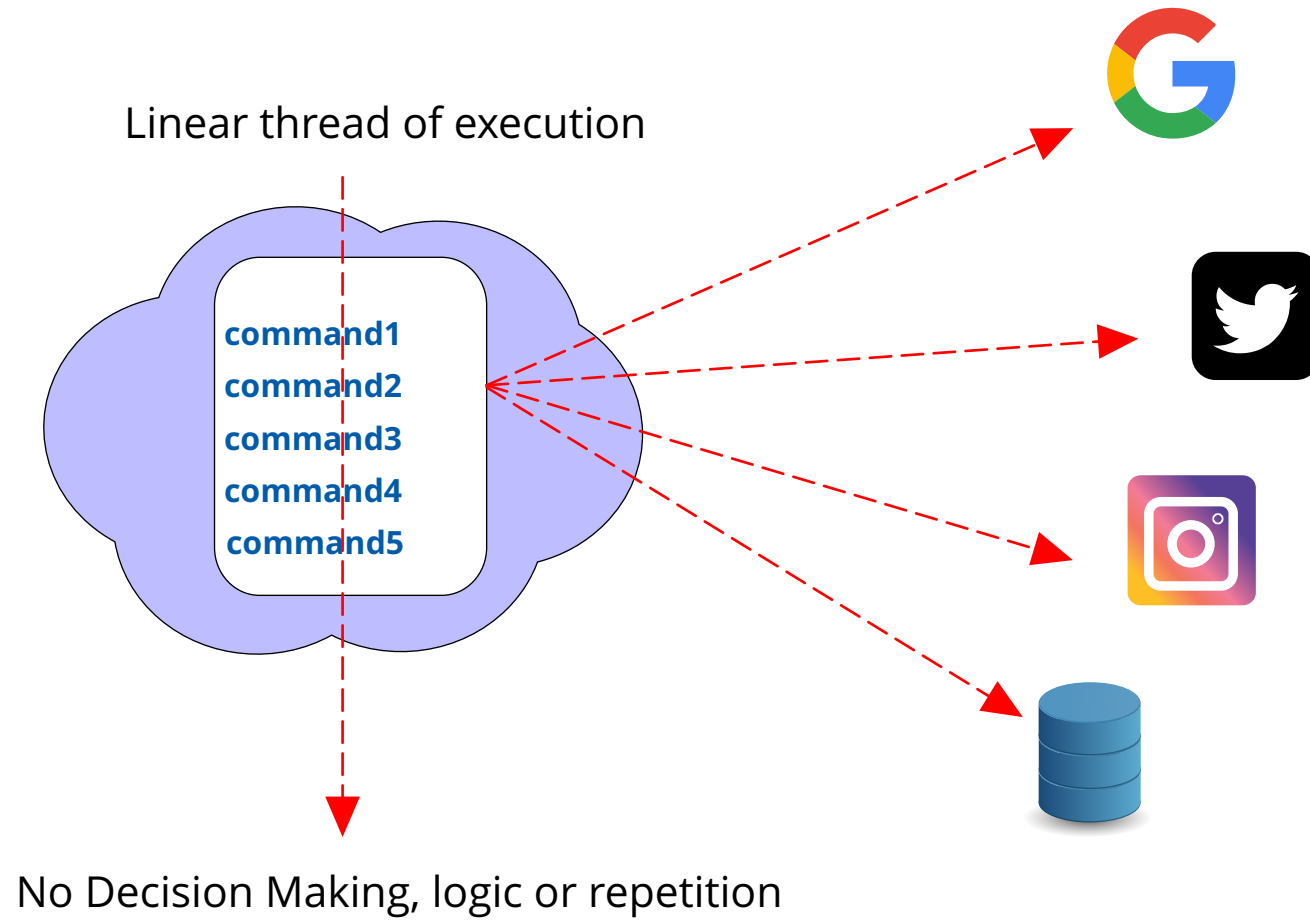


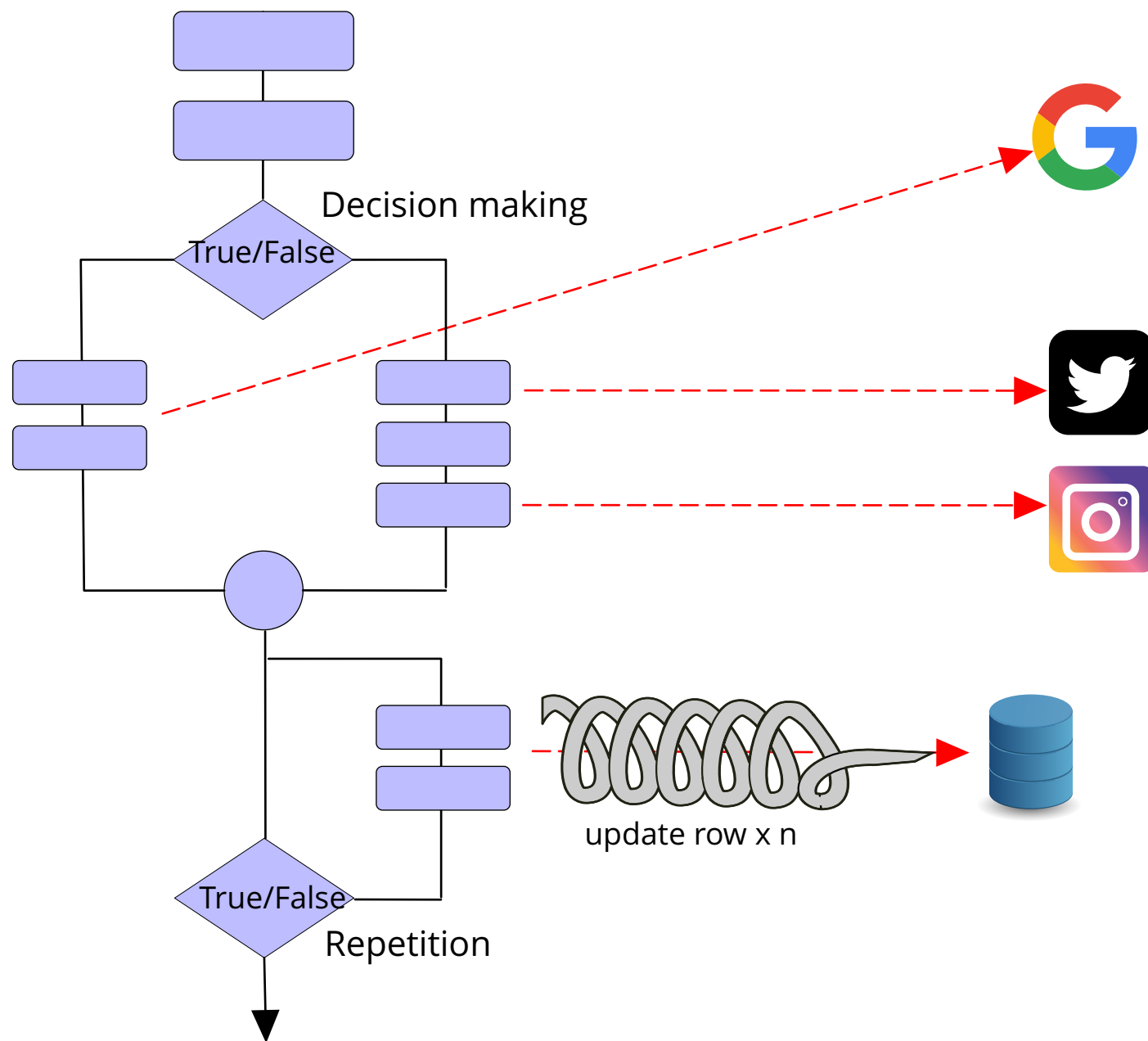
```
>>> capitals['Germany'] = 'Berlin'
```

```
>>> print(capitals['France'])
```



Use PEP 008 guidelines when writing GOOD Python code!  
Details layout of code including space, tabs, indentations etc

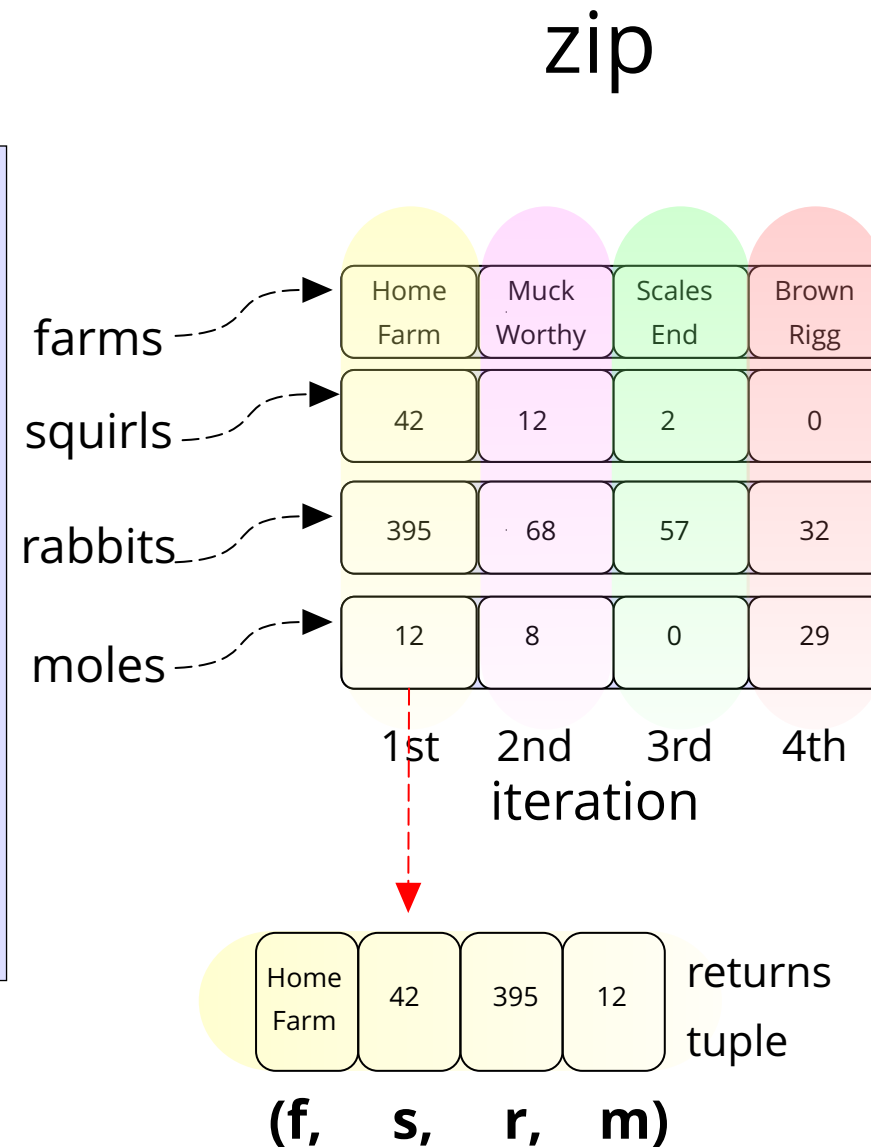




```
#!/usr/bin/python
# Single Line Comment

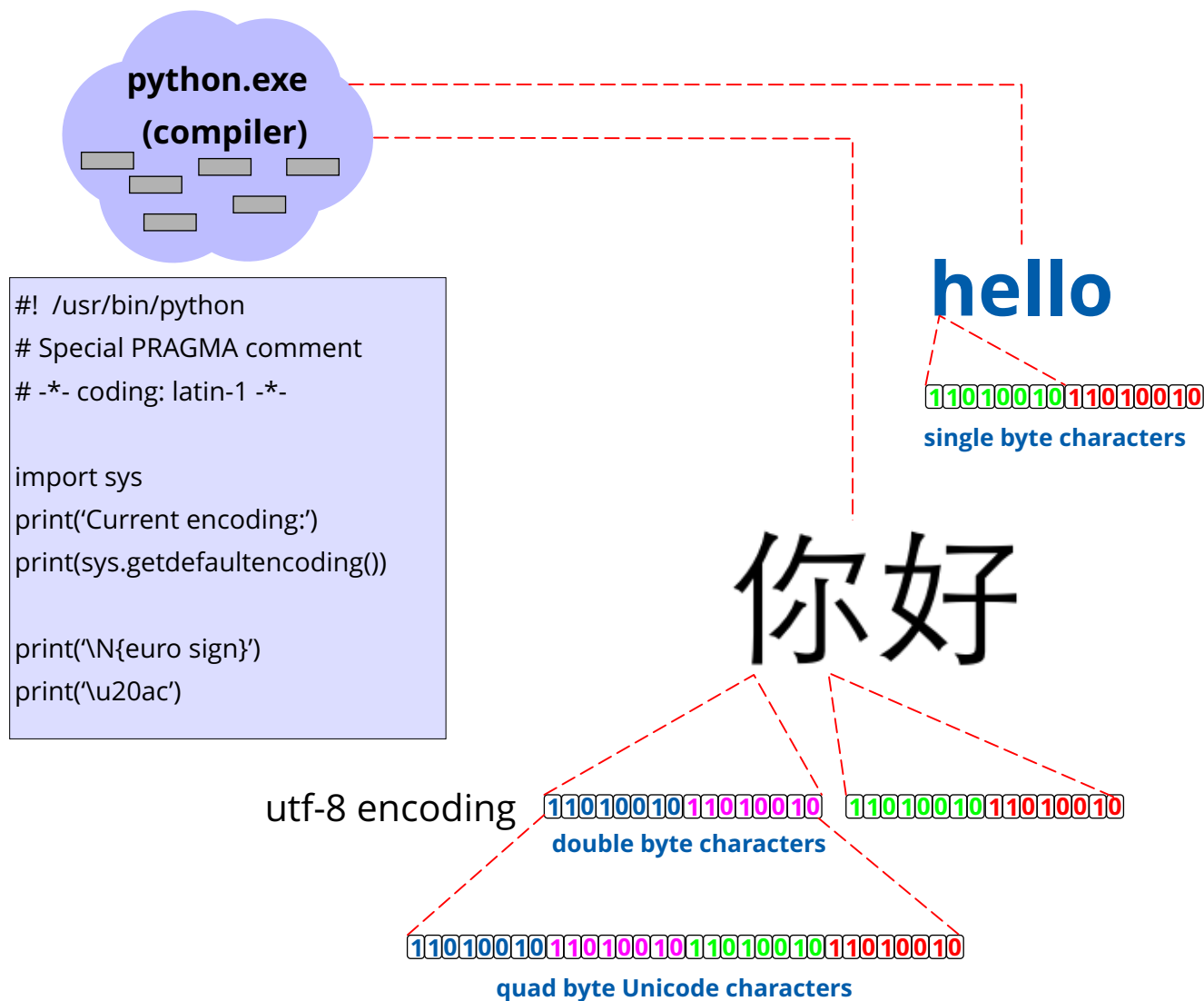
farms = ['Home Farm', 'Muckworthy',
         'Scales End', 'Brown Rigg']
squirrels = [42, 12, 2, 0]
rabbits = [395, 68, 57, 32]
moles = [12, 8, 0, 29]

for f, s, r, m in zip(farms, squirrels, 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....



```
>>> 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

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000	b	B	Ĭ	Ĭ	Ĭ	Ĭ	Ĭ	Ĭ	Ĭ	Ĭ	Ĭ	Ĭ	Ĭ	Ĭ	Ĭ	Ĭ
030	Ĭ	F	f	Ĭ	Y	hu	l	i	K	k	f	l	u	N	ŋ	Ō
060	Ō	σ	Ql	q	P	β	R	z	z	Σ	l	l	T	f	T	U
090	u	Ō	U	Y	y	Z	z	3	Ĭ	z	z	2	S	z	s	p
120	l	l	+	!	DŽ	Dz	dž	Lj	Lj	Nj	Nj	Nj	nj	Ĭ	a	Ĭ
150	Ĭ	Ō	o	Ō	u	Ō	u	Ō	u	Ō	u	Ō	u	o	Ĭ	ā
180	Ĭ	ā	Ĭ	æ	G	g	Ĭ	g	K	k	Q	q	Ō	Ō	3	3
210	J	DZ	Dz	dz	Ĭ	g	Hu	P	N	n	Ĭ	ā	Ĭ	æ	æ	Ō
240	Ĭ	ā	Ĭ	a	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	Ĭ	Ō	Ō	Ō	Ō
270	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z
300	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z
330	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z
360	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z
390	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z
420	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z
450	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z
480	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z
510	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z
540	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z
570	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z
600	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z	Ĭ	z

Latin Extended-B

[Click to separate range](#)  
 Range: 0180—024F  
[Click to highlight range](#)  
 N:   
 Languages: slovenian, croatian





Py3

positional arguments      named arguments

```
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")
```



```
#!/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)+'>')
```

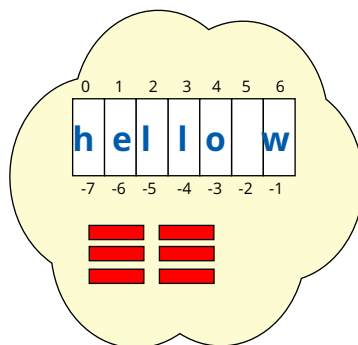
```
Hello  
HELLO  
<  hello  >  
<hello    >  
<      hello>  
<0000000hello>
```

```
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
```

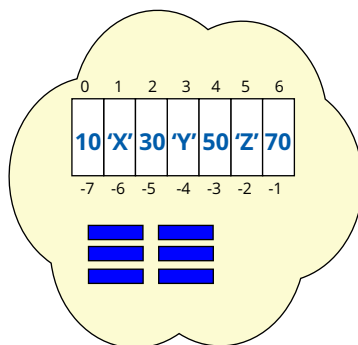
```
"{0:2d} {1:<10s} {2:06.2f} Gm".format(i, key, planets[key])
```

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

**str**

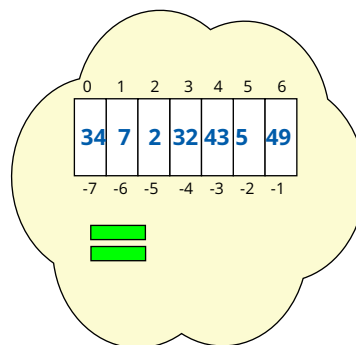
**Ordered**  
**Indexed using [n]**  
**Immutable**

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

**list**

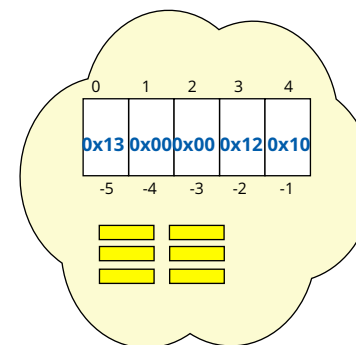
**Ordered**  
**Indexed using [n]**  
**Mutable**  
**Dynamic, flexible**

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

**tuple**

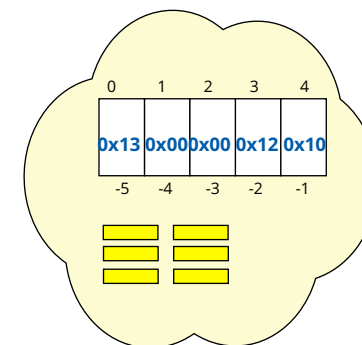
**Ordered**  
**Indexed using [n]**  
**Immutable**  
**Fast, simple**

```
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**

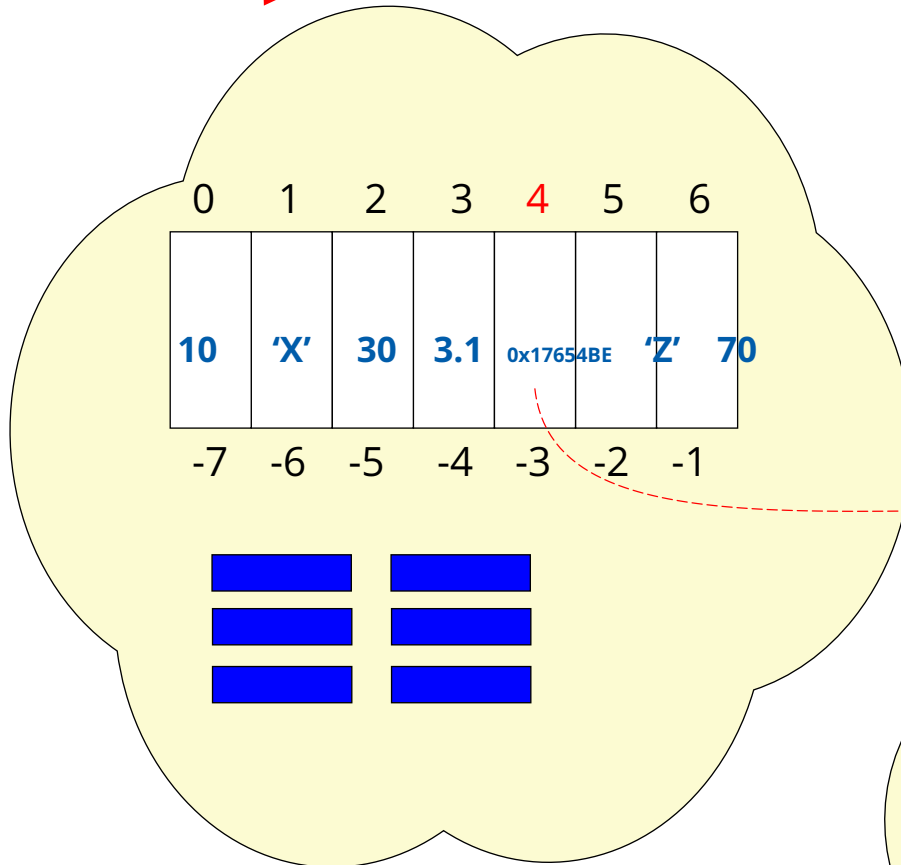
```
mybytes = bytes('hello', 'utf-8')
mybytes = b'hello'
mybytes.isalpha()
mybytes.decode()
mybytes[-1]
```



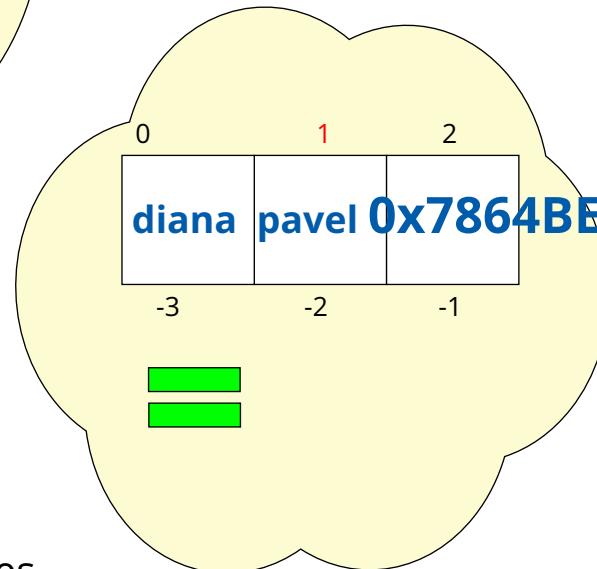
**mylist[4][1]['home']**  
**mylist**

**0x12345FF**

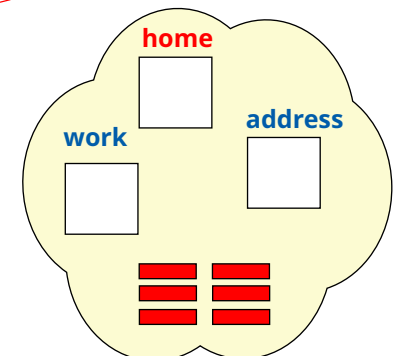
**mylist = [10,'x',30,3.1,['diana','pavel',{'home':'0141'  
'work':'0875'}], 'z',70]**



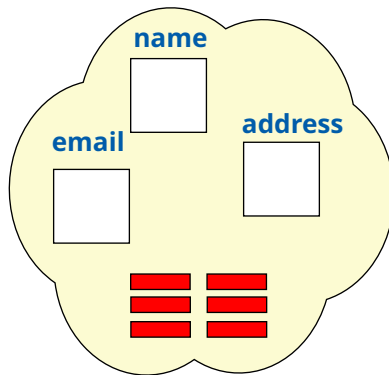
**0x17654BE**



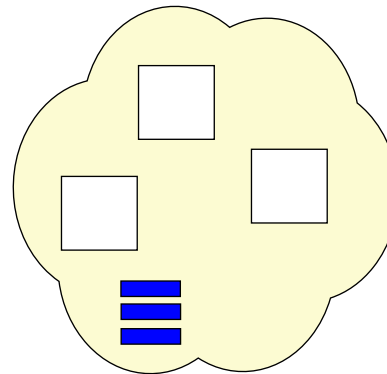
**0x7864BE**



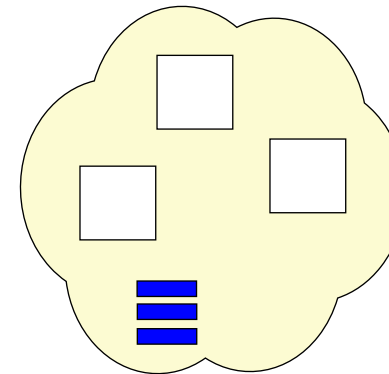
Use Pretty Print (import pprint)  
module to visualise complex structures

**dict**

**Unordered**  
**Indexed using ['key']**  
**Mutable**  
**Dynamic, fast searching**  
**Keys must be unique**

**set**

**Unordered**  
**Unique data values**  
**Mutable**

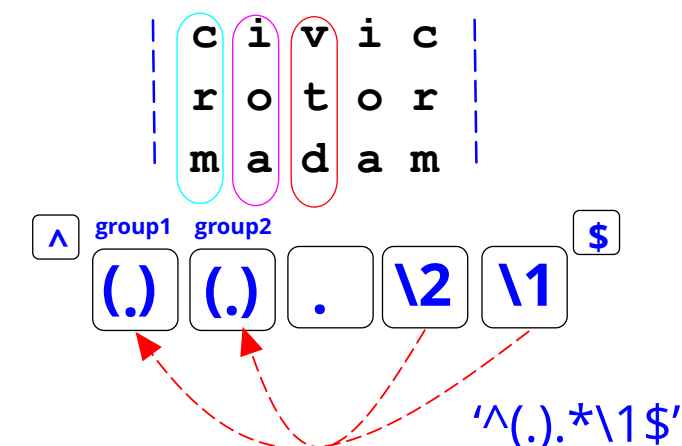
**frozenset**

**Unordered**  
**Unique data values**  
**Immutable**

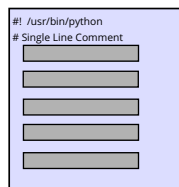


B.R.E/Regex/E.R.E

**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}$`UNIX: `$ grep 'pattern$' file1 file2 *`**Escape Char**`'\.'` = escape next char**Repetition Char**`'e*'` = repeat 0 or more

`'[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**Alternation (or)**`'donaldd | dcameron | Sir Donald'`**Groupings/Back Referrals****Python Escape/Assertions**`\d` - [0-9]`\w` - [a-zA-Z0-9\_]`\s` - [\t\n\r\f]`\b` - [\t;,:?!"'<>]`\A` - start of string`\D` - [^0-9]`\W` - [^a-zA-Z0-9\_]`\S` - [^ \t\n\r\f]`\B` - [^ \t;,:?!"'<>]`\Z` - end of string`:+-15digits:``'[:+]? \d{15}:'`

## 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")
```

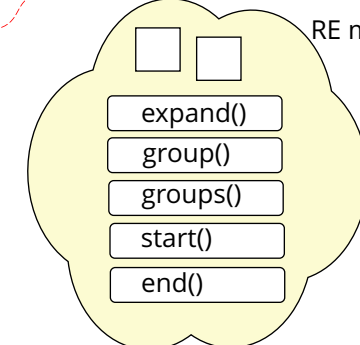
```
print(m.group())
```

```
print(m.groups())
```

```
print(m.groups()[0])
```

```
print(m.group(1))
```

m → 0x1234abcd



RE matchobject

consider pre-compiling pattern

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

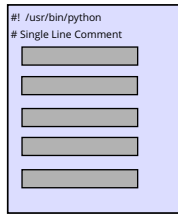
```
reobj = re.compile(r"^(root).*\1")
for somethings in biglist:
    m = reobj.search(something)
```

returns matched string

returns tuple of matched group members

returns match from group 1 (yuck!)

returns match from group 1 (Pythonic)



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 <i>line</i>
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

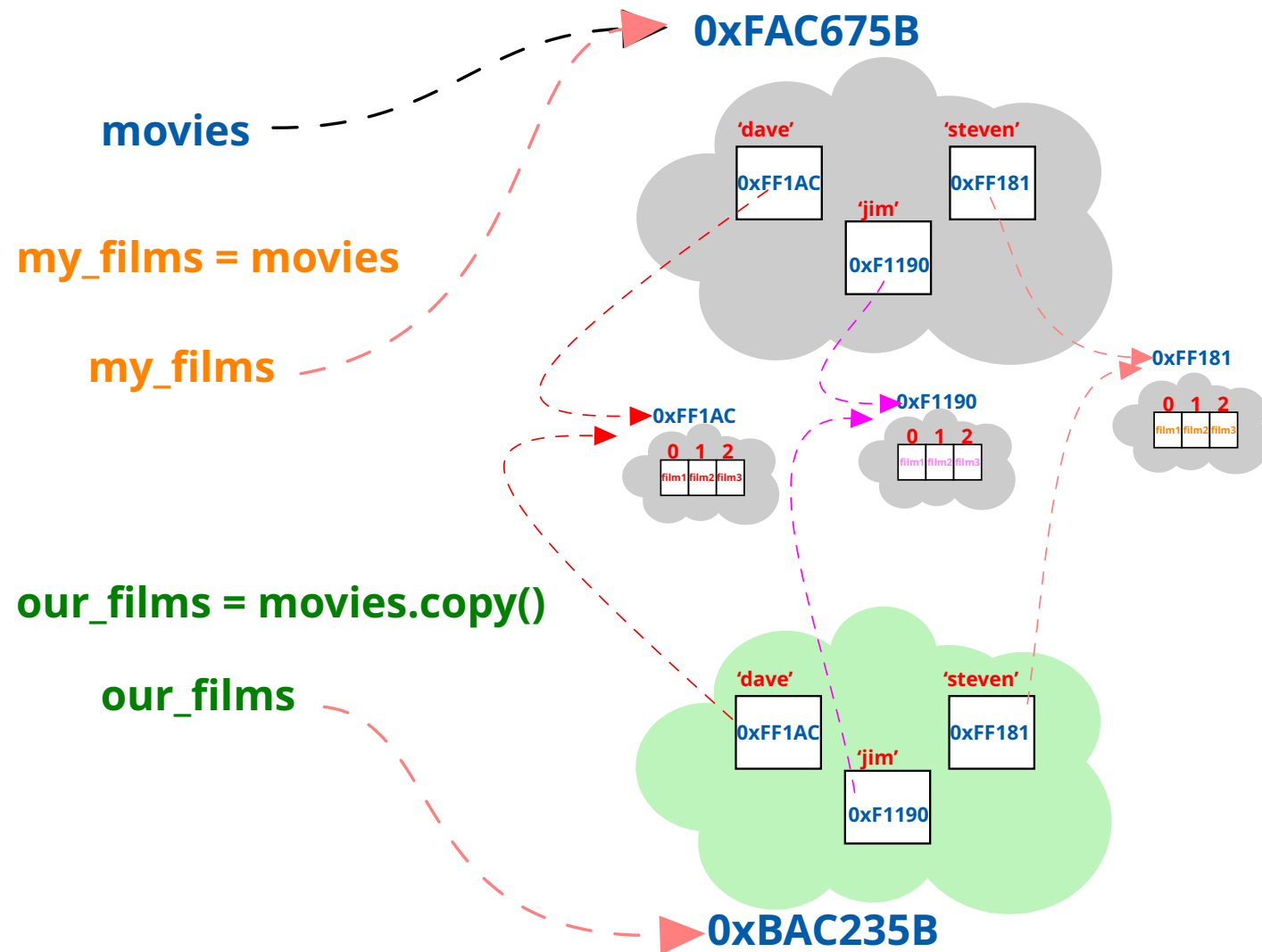
m = re.match(r"(root).\*\1", line, flags=re.IGNORECASE | re.MULTILINE)

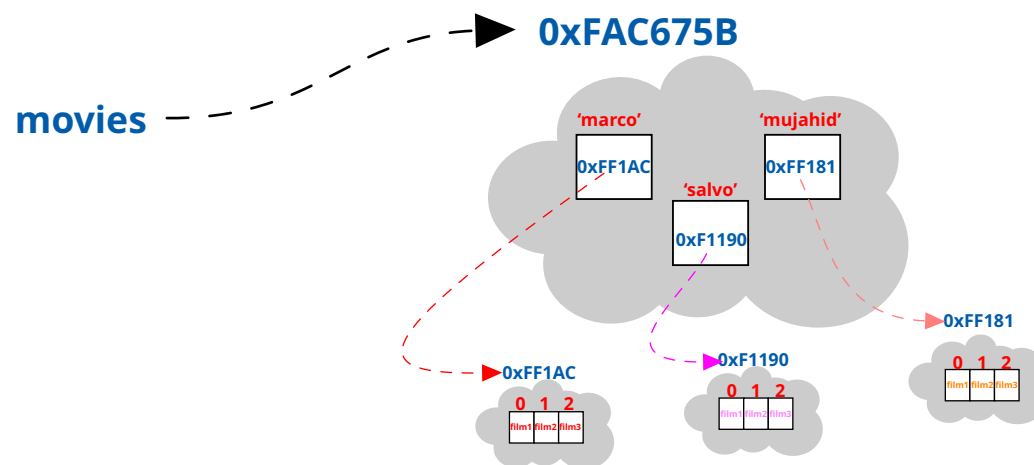
m = re.fullmatch(r"^root.\*\$", line)

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

modifier span (Py3.6)

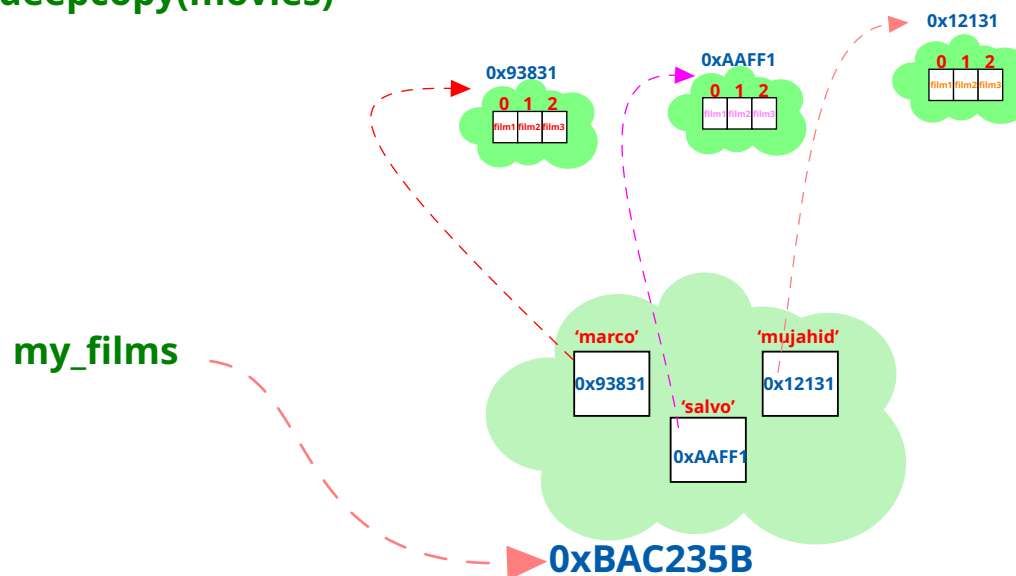






```
import copy
```

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



```
import sys
```

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

C:\

pip/Pycharm/IDE

program\_files

Python36-32

\$HOME

import\_modules

python

DLLs

lib

\_\_pycache\_\_

Module/s

Timestamp == Timestamp

Package = logical group  
of modules in a dir

COD

project2

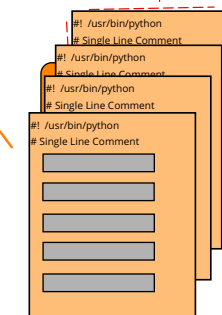
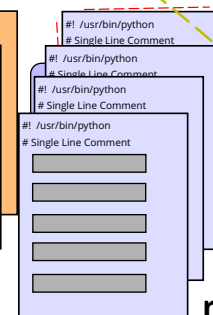
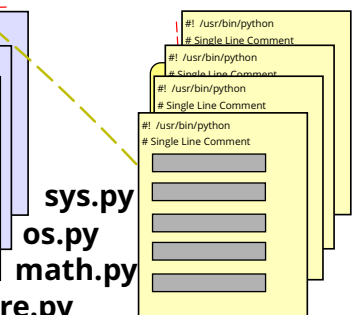
graphics audio logic tanks buildings

mod1.py mod2.py .py .py .py

Typically - each module will have 1 class inside

\_\_init\_\_.py

all = ['mod1','mod2']

Pre-compiled  
C programs .pydPython  
Standard  
Librarysys.pyc  
os.pyc  
math.pyc  
re.pyc  
Pre-compiled  
.pysys.py  
os.py  
math.py  
re.py

Determines  
which modules  
can be imported from  
this package

from COD import \*

Python programs == Module

Names should be in short/lowercase (PEP008)

## \$PYTHONPATH

```
import sys
```

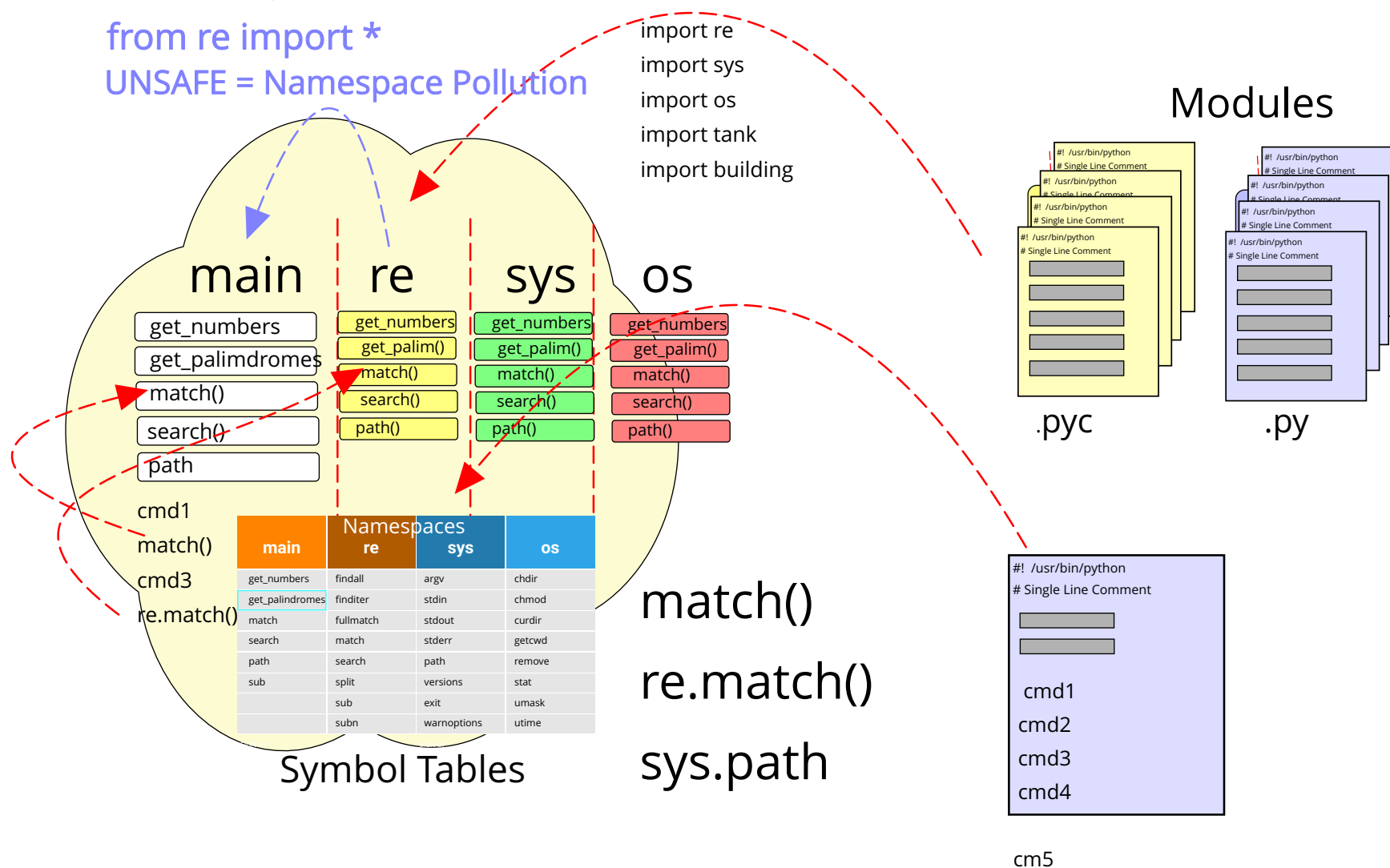
```
sys.path.append('C:\import_modules')
```

```
from re import search, match
```

```
from re import *
```

## UNSAFE = Namespace Pollution

```
import re
import sys
import os
import tank
import building
```



\$PYTHONPATH

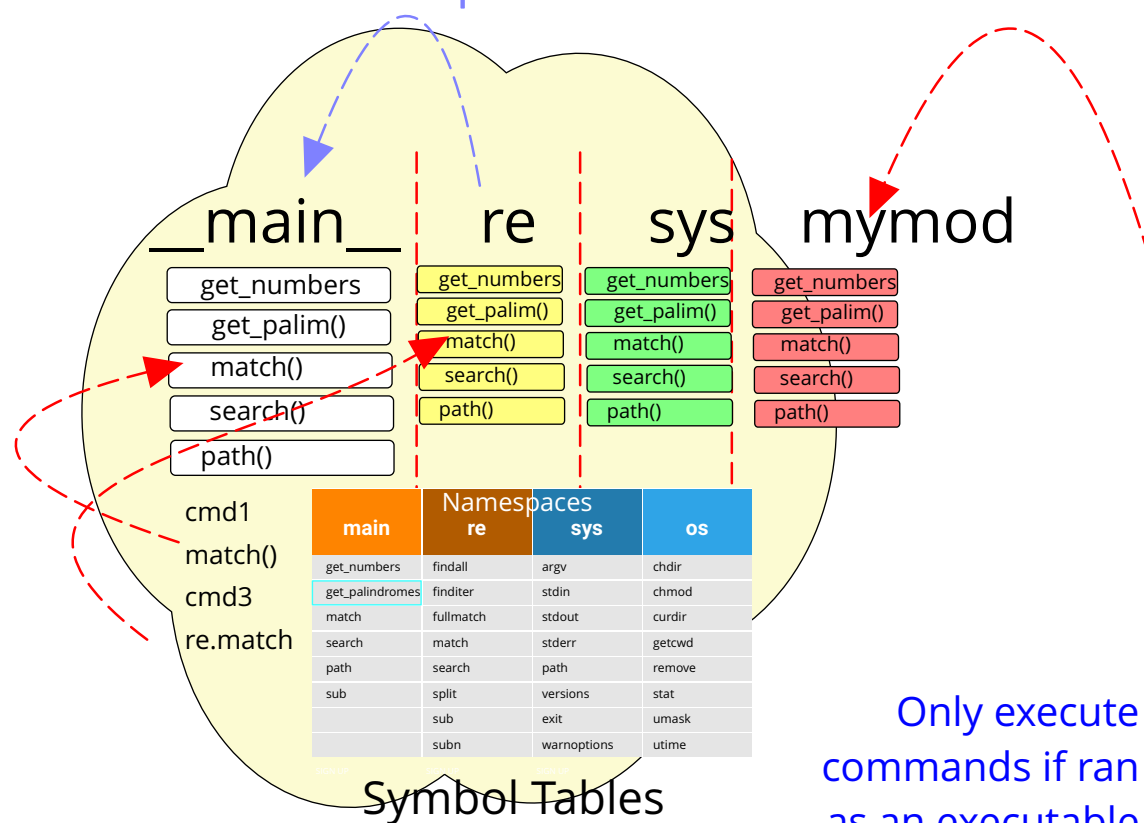
import sys

sys.path.append('C:\import\_modules')

mymod

from re import search, match  
from re import \*  
UNSAFE = Namespace Pollution

import mymod



Only execute  
commands if ran  
as an executable

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

```
def func1():
```

```
-----
```

```
def func2():
```

```
-----
```

```
def main():
```

```
    cmd1
```

```
    cmd2
```

```
    cmd3
```

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



tree1 = type height  
 tree2 = colour XYZ  
 tree3 =    
 tree4 =    
 building1 =    
 building2 =  XYZ  
 building3 =    
 tank1 = country type  
 tank2 = speed XYZ  
 tank3 =

Thing

Thing

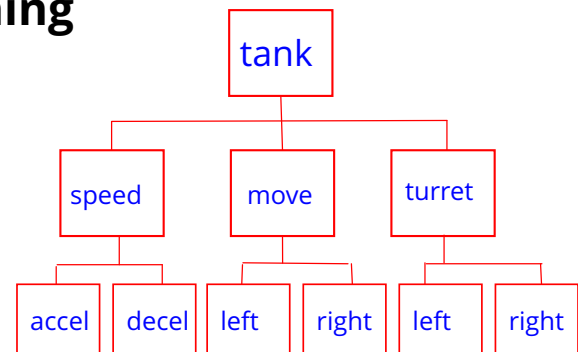
Thing

tank  
data

```

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()
    }

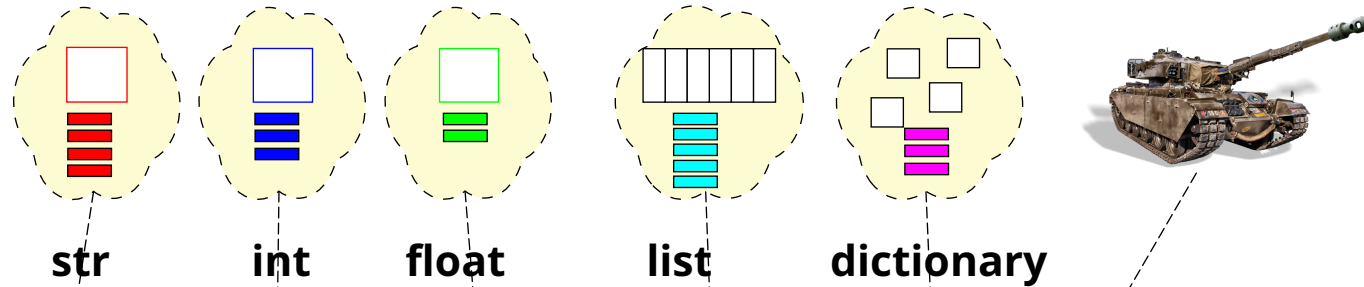
```

tank  
behaviourstructured/modular  
programmingreplicate for every  
tree, building, rock,  
soldier and tank...

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



## Basic built-in classes



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

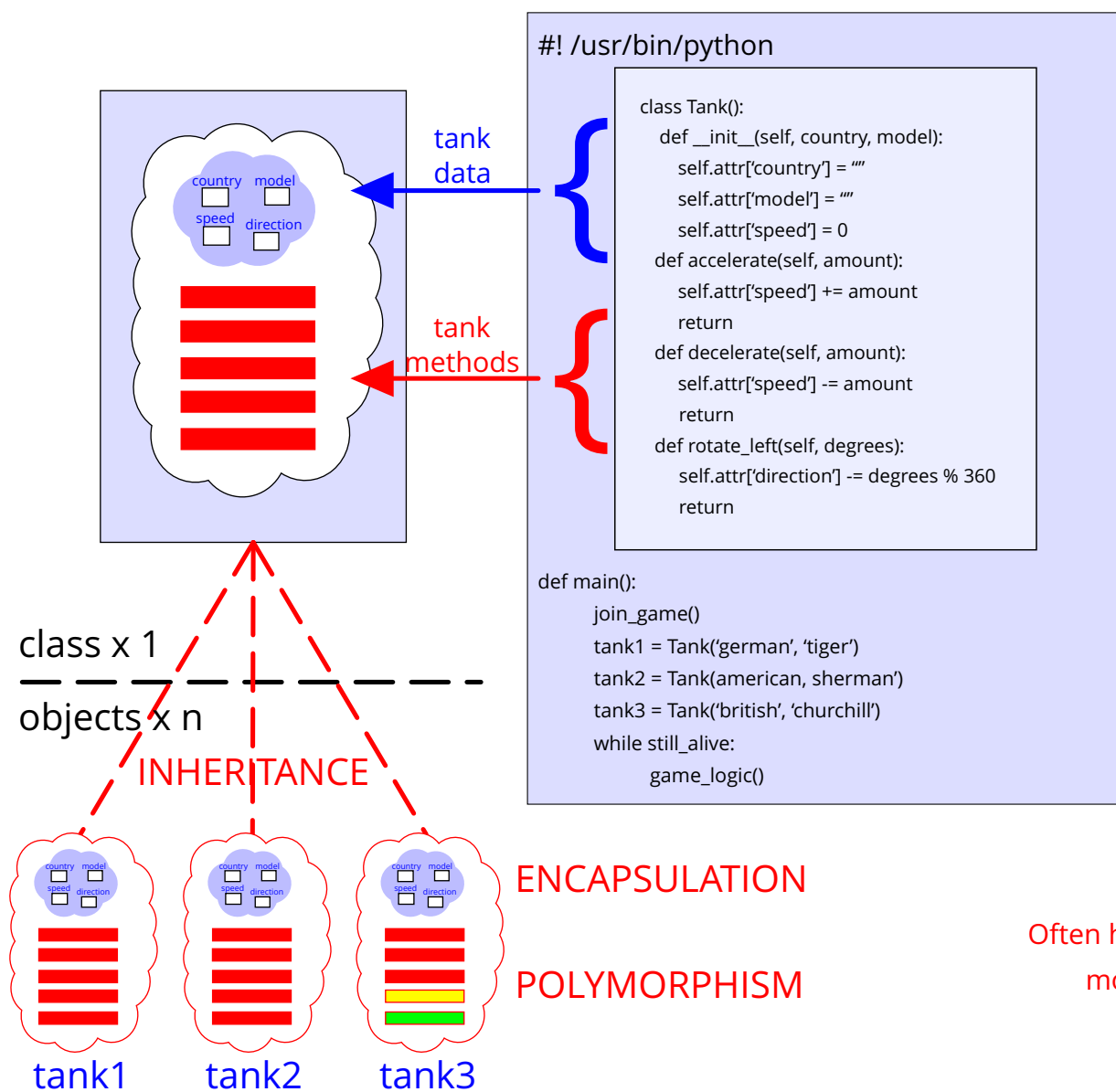
```
print(str, int, float, list, dictionary, Tank)
```



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

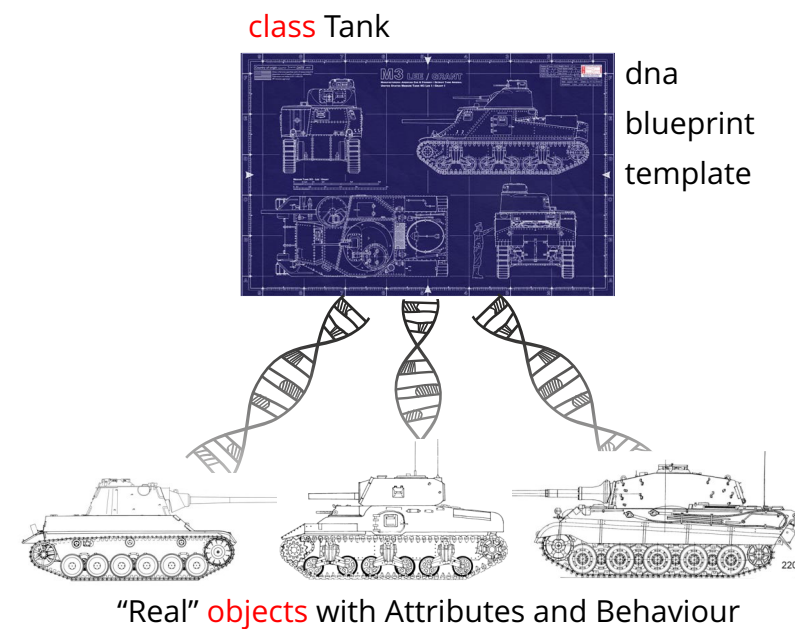
*If it looks like a duck,  
swims like a duck,  
and quacks like a duck,  
then it probably is a duck*

Objects methods  
define what it can do  
Not its type!

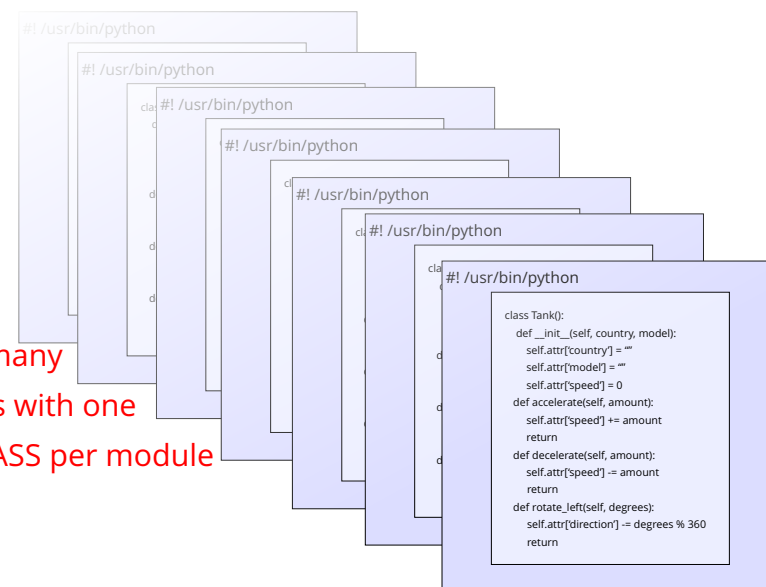


Created @ runtime

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



Often have many  
modules with one  
CLASS per module





```
#!/usr/bin/python
```

```
# comments
```

```
"""
```

```
    This is a docstring describing what  
    the program/module/class does
```

```
"""
```

```
import sys
```

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

```
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"Successfully opened file {filename}")
```

```
BaseException  
+-- SystemExit  
+-- KeyboardInterrupt  
+-- GeneratorExit  
+-- Exception  
+-- StopIteration  
+-- StopAsyncIteration  
+-- ArithmeticError  
+-- FloatingPointError  
+-- OverflowError  
+-- ZeroDivisionError  
+-- AssertionError  
+-- AttributeError  
+-- BufferError  
+-- EOFError  
+-- ImportError  
+-- ModuleNotFoundError  
+-- LookupError  
+-- IndexError  
+-- KeyError  
+-- MemoryError  
+-- NameError  
+-- UnboundLocalError
```

```
+-- Exception  
+-- OSError  
+-- BlockingIOError  
+-- ChildProcessError  
+-- ConnectionError  
+-- BrokenPipeError  
+-- ConnectionAbortedError  
+-- ConnectionRefusedError  
+-- ConnectionResetError  
+-- FileExistsError  
+-- FileNotFoundError  
+-- InterruptedError  
+-- IsADirectoryError  
+-- NotADirectoryError  
+-- PermissionError  
+-- ProcessLookupError  
+-- TimeoutError
```

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

Exceptions are classes,  
and have a hierarchy

```
#!/usr/bin/python
# comments main thread
"""
    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)
```

Thread 1

Thread 2

