



DIEE - Università degli Studi di Cagliari

# OOP and Scripting in Python

## *Part 1 - Basic Features*

Giuliano Armano – DIEE Univ. di Cagliari



---

# Part 1 - Basic Features



# Python: Basic Features

---

- Data Structures
- Control Structures
- Functions
- Polymorphic variables
- Garbage Collection



---

# Data Structures

## Part 1 - Basic Features: Data Structures

Giuliano Armano


1



# Data Structures (conventional)

---


- Numbers (int, long, double) YES
- Booleans YES
- Characters NO
- Strings YES
- Arrays NO
- Records NO



# Numbers

---


```
>>> x = 1                # int
>>> print x
1
>>> y = 10L              # long
>>> print y
10
>>> z = x + 2.3          # double
>>> print z
3.3
>>> print x+z
4.3
```



# Booleans

---

```
>>> print True or False
True
>>> print True and False
False
>>>
```



# Strings

*Strings are immutable*

---

```
>>> print "Hello, world!"  
Hello, world!  
>>> print 'Hello, world!'  
Hello, world!  
>>> a = "blob"  
>>> b = a  
>>> a += '-one'  
>>> print a,b  
blob-one blob  
>>>
```





# Data Structures (advanced)

---

- Big numbers YES
- Complex numbers YES
- Lists YES
- Tuples YES
- Hash tables (dictionaries) YES

```
>>> fact(200)
```

>>>



# Complex Numbers

---

```
>>> c = complex(1.,2.)
```

```
>>> c.real
```

```
1.0
```

```
>>> c.imag
```

```
2.0
```

```
>>> c
```


```
(1+2j)
```

```
>>> c += (3+4j)
```

```
>>> c
```

```
(4+6j)
```

```
>>>
```



# Lists

---

```
>>> L = ['a','b','c']
```

```
>>> L
```

```
['a', 'b', 'c']
```

```
>>> L += [1,2]
```

```
>>> L
```

```
['a', 'b', 'c', 1, 2]
```

```
>>> L[2]
```

```
'c'
```

```
>>> L[-1]
```

```
2
```

```
>>> list('blob')
```

```
['b', 'l', 'o', 'b']
```

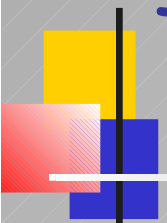
```
>>> list('blob')[2]
```

```
'o'
```

```
>>> list('blob'+ 's')
```

```
['b', 'l', 'o', 'b', 's']
```

```
>>>
```



# Tuples

*Tuples are immutable*

---

```
>>> T = ('x', 'y', 'z')
```

```
>>> T
```

```
('x', 'y', 'z')
```

```
>>> T[0]
```

```
'x'
```

```
>>> T[0:2]
```

```
('x', 'y')
```

```
>>> T[2]
```

```
'z'
```

```
>>> T[-2]
```

```
'y'
```

```
>>>
```

```
L = list(T)
```

```
>>> L += ['w']
```

```
>>> L
```

```
['x', 'y', 'z', 'w']
```

```
>>> T1 = tuple(L)
```

```
>>> T
```

```
('x', 'y', 'z')
```

```
>>> T1
```

```
('x', 'y', 'z', 'w')
```

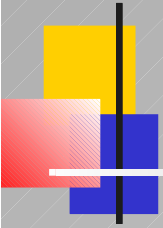
```
>>>
```



# Dictionaries

---

```
>>> D = {'john' : 32, 'paul' : 44, 'helen' : 38}
>>> D
{'paul': 44, 'john': 32, 'helen': 38}
>>> D['john']
32
>>> D['helen']
38
>>> D['helen'] = 37
>>> D
{'paul': 44, 'john': 32, 'helen': 37}
>>> dict([('x',1), ('y',2), ('z',3)])
{'y': 2, 'x': 1, 'z': 3}
>>>
```



# Control Structures

## Part 1 - Basic Features: Control Structures

Giuliano Armano

1

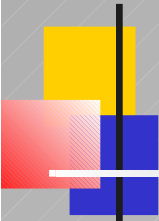


# Control Structures

---

- if\_then / if\_then\_else YES
- case (using if\_then\_elif\_else) YES
- while\_do YES
- for YES
- Omega-N (using while\_do + break) YES





# if\_then / if\_then\_else

---

```
>>> if x < 100: print 'hello'      # if_then
...

hello
>>>
>>> x = 40
>>> if x == 40: print 'foo'        # if_then_else
... else: print 'oof'
...

foo
>>>
```



# Case

---

```
>>> y = 3
>>> if y==1: print 'one'
... elif y==2: print 'two'
... elif y==3: print 'three'
... else: print 'unknown'
...
```

three

```
>>>
```



# while\_do

---

```
>>> x = 5
>>> while x > 0:
...     print 'foo', x
...     x -= 1
...

foo 5
foo 4
foo 3
foo 2
foo 1
>>>
```



# For (e.g. on an integer range)

---

```
>>> range(5)
[0, 1, 2, 3, 4]
>>> for x in range(5):
...     print 'foo', x
...

foo 0
foo 1
foo 2
foo 3
foo 4
>>>
```



# For (e.g. on a string)

---

```
>>> for x in "blob":  
...     print 'foo', x  
...
```

```
foo b
```

```
foo l
```

```
foo o
```

```
foo b
```

```
>>>
```



# For (e.g. on a list)

---

```
>>> for x in ['a','b','c','d']:  
...     print 'foo', x  
...
```

```
foo a
```

```
foo b
```

```
foo c
```

```
foo d
```

```
>>>
```



# For (e.g. on a tuple)

---

```
>>> for x in ('a','b','c','d'):  
...     print 'foo', x  
...
```

```
foo a
```

```
foo b
```

```
foo c
```

```
foo d
```

```
>>>
```



# For (e.g. on a dictionary)

---

```
>>> d = {'a':1, 'b':2, 'c':3}
```

```
>>>
```

```
>>> for x,y in d.items():
```

```
...     print 'foo', x,y
```

```
...
```

```
foo a 1
```

```
foo c 3
```

```
foo b 2
```

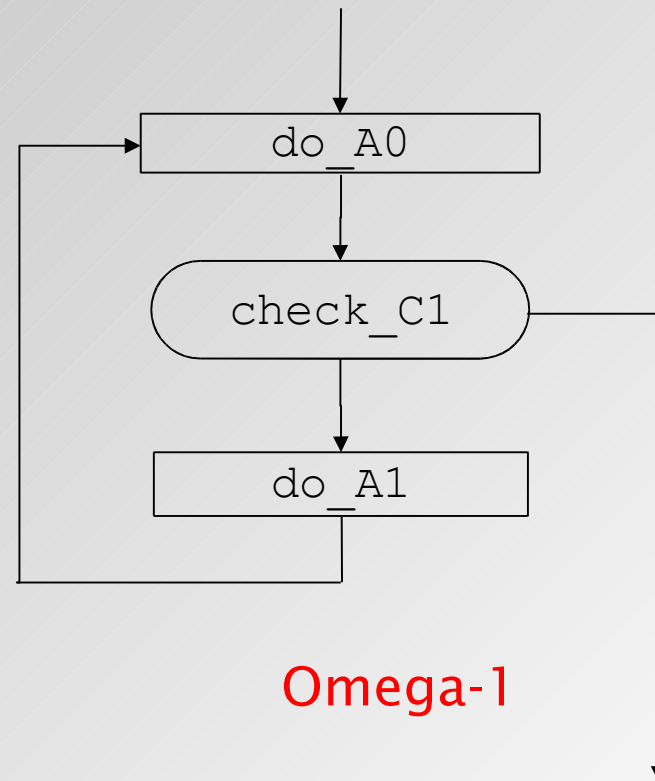
```
>>>
```



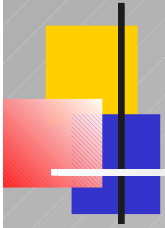
# Omega-N

```
>>> C1 = 2
>>> while True: # omega-1
...     print 'doing A0'
...     if C1 <= 0: break
...     print 'doing A1'
...     C1 -= 1
... 
```

```
doing A0
doing A1
doing A0
doing A1
doing A0
>>> 
```



Omega-1



# Functions

## Part 1 - Basic Features: Functions

Giuliano Armano



# Functions

---

- Arguments' type checking **NO**
- Default parameters **YES**
- Keyworded parameters **YES**
- Apply functions to arglists **YES**



# Arguments' Type Checking

---

```
>>> def foo(x):  
...     print x  
...
```

```
>>> foo('blob')  
blob
```

```
>>> foo(34)  
34
```



# Default parameters

---

```
>>> x1 = 0
>>> def foo(x=x1):
...     print x
...

>>> foo(10)
10
>>> foo()
0
>>> x1 = 33
>>> foo()      # 0 or 33 ? 0
0
>>>
```



# Keyworded parameters

---

```
>>> def foo(x=0,y=0,z=0):  
...     print "x,y,z = ",x,y,z  
...
```

```
>>> foo()
```

```
x,y,z = 0 0 0
```

```
>>> foo(11)
```

```
x,y,z = 11 0 0
```

```
>>> foo(y=33)
```

```
x,y,z = 0 33 0
```

```
>>> foo(z=1,y=33)
```

```
x,y,z = 0 33 1
```

```
>>>
```



# Apply functions to arglists

---

```
>>> def foo(x=0,y=0,z=0):  
...     print "x,y,z = ",x,y,z  
...
```

```
>>> apply(foo,[1,2,3])           # deprecated
```

```
x,y,z = 1 2 3
```

```
>>> foo(*[1,2,3])                # OK, new programming style
```

```
x,y,z = 1 2 3
```

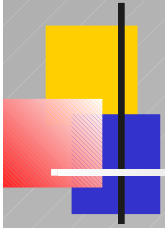
```
>>> foo(*[10,20])                # z=0 (default value)
```

```
x,y,z = 10 20 0
```

```
>>> foo(**{'x':1, 'z':3})        # y=0 (default value)
```

```
x,y,z = 1 0 3
```

```
>>>
```



# Polymorphic Variables

## Part 1 - Basic Features: Polymorphic Variables

Giuliano Armano

1

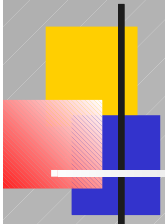




# Polymorphic Variables

---

- Type declaration (for variables) **NO**
- Type checking (on assignment) **NO**
- Scope (lexical vs. dynamic) **DYNAMIC**
- Extent (automatic vs. dynamic) **DYNAMIC**

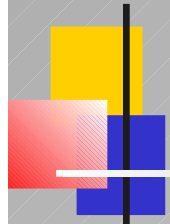


# Polymorphic Variables

*Type Declaration and Type Checking*

---

```
>>> def foo(x):  
...     print x  
...  
  
>>> a = 1  
>>> a  
1  
>>> a = "blob"  
>>> a  
blob  
>>> a = foo  
>>> a  
<function foo at 0x00A9E170>
```



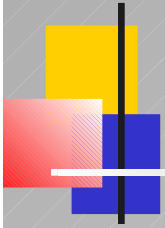
# Polymorphic Variables

*Dynamic Scope and Extent*

---

```
>>> x = 35
>>> def foo():
...     print x
...

>>> foo()
35
>>>
```



# Garbage Collection

## Part 1 - Basic Features: Garbage Collection

Giuliano Armano

1



# Garbage Collection

---

- No need for object destructors
- Unused values / objects are collected and removed by a suitable algorithm (= distributed garbage collector)