Scripting & Computer Environments

Core Python: Control Structures

IIIT-H

Oct 23, 2013

...Previously & Today...

Previously: Sequence objects (a.k.a sequences)

- `Strings`
- `Lists`

Today:

- More core objects
  1. `Tuples` (sequences)
  2. `Dictionaries` (mappings)
- Flow control statements

Brainstorm

1. Mutable vs Immutable Objects?
2. In Python, when is a variable created? What is its type?
3. How do you copy values to variables?

Brainstorm

1. Mutable vs Immutable Objects?
2. In Python, when is a variable created? What is its type?
3. How do you copy values to variables?

Brainstorm

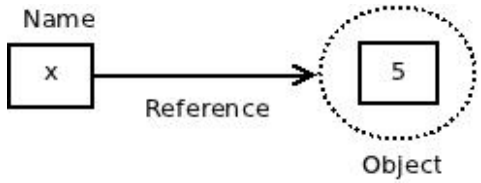
1. Mutable vs Immutable Objects?

2. In Python, when is a variable created? What is its type?

3. How do you copy values to variables?

Object References

- Variables link to objects. The links are a.k.a. *references*.



- Assignment statements do NOT copy objects.
- They manipulate object references/bindings.

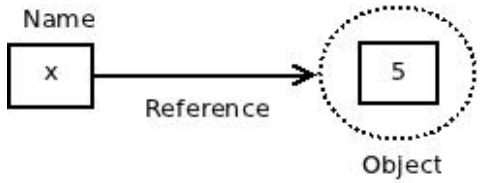
Examples

```
>>>x=5
>>>y=x
>>>x='hola'
>>>y
```

```
>>>print id(x)
>>>print id(y)
>>>print id(y)
>>>print id(x)
```

Object References

- Variables link to objects. The links are a.k.a. *references*.



- Assignment statements do NOT copy objects.
- They manipulate object references/bindings.

Examples

```
>>>x=5                                    >>>print id(x)
>>>y=x                                    >>>print id(y)
>>>x='hola'                               >>>print id(y)
>>>y                                      >>>print id(x)
```

Example

```
>>>dept = [‘Bioinformatics’, ‘CSE’]

>>>x = dept

>>>x = ["VLSI", "CL"]

>>>print dept

>>>x = dept

>>>x[0] = ‘PGSSP’

>>>x.append(‘C-STAR’)

>>>print dept
```

Example

```
>>>s = "CAGTTGGGACTAG"

>>>r = "AGTC"

>>>n = s.count('G')

>>>print n

>>>r = s                    # what happens to 'AGTC'?

>>>s = "ACGAT"

>>>print r

>>>del r

>>>print r
```

Sequence Objects: 3. Tuples

- Ordered set of items inside `()` - optional parenthesis.
- Are *immutable* sequences, fixed-size.

```
>>>T=()
>>>T=(4,)                    # notice the comma
>>>T=(4,(6,8))               # are nestable
```

- Offset-based access.

```
>>>T[0]
>>>T[-1]                  # negative-indexing
>>>T[99]
```

- Hetrogeneous collection

Tuple Operations

- Concatenation (+)

```
>>>T=(2,4) + (3,4)
```

- Repetetion (*)

```
>>>T * 3
```

- Indexing

- Slicing

```
>>>T[:-1]
>>>T[1:2]
```

- Assignment

```
>>>(a,b,c)=(1,2,3)
# size matters
```

Tuple Methods

- `len(T)`
- `T.count(elem)`
- `T.index(elem)`
- `sorted(T)  # a list`

Any other way of sorting tuples?

Tuple Methods

- `len(T)`
- `T.count(elem)`
- `T.index(elem)`
- `sorted(T)  # a list`

Any other way of sorting tuples?

```
>>>tmp=list(T)     # convert to a mutable type
>>>tmp.sort()
>>>T=tuple(tmp)
```

Mapping Objects: Dictionaries

- *Unordered contrainers* that map/bind keys to values.

```
dict = {key1:value1, key2:value2, key3:value3...}
```

- i.e. set of *unordered* key-value pairs.
- a.k.a. *hashes/associative arrays*.
- Key-based access, and not positional.
- Hetrogeneous, nestable
- Mutable (can shrink and grow)

Table: Greetings

Key	Value
English	Hello
Hindi	Namaste
Spanish	Hola
French	Bonjour
Amharic	Selam

```
>>>Greetings = { "English":"Hello", "Hindi":"Namaste", "Spanish":"Hola", \
                 "French":"Bonjour", "Amharic":"Selam" }


>>>print Greetings["English"]


>>>print Greetings["HINDI"]                # case-sensitive
```

Dictionary Operations

- Construction

```
>>>D={}                    # empty dictionary

>>>D={‘title’:‘Monty Python’,‘genre’:‘comedy’}

>>>D= { (1,2,3):"hi", 4:"bye" }              # Key be immutable

>>>D={ [1,2,3]:"hi", 4:"bye"}                # Error!
```

- Lookup

```
>>>D[‘title’]

>>>D[‘year’]                          # ‘KeyError’ exception
```

Dictionary Operations

- Construction

```
>>>D={}                    # empty dictionary

>>>D={‘title’:‘Monty Python’,‘genre’:‘comedy’}

>>>D= { (1,2,3):"hi", 4:"bye" }              # Key be immutable

>>>D={ [1,2,3]:"hi", 4:"bye"}                # Error!
```

- Lookup

```
>>>D[‘title’]

>>>D[‘year’]                          # ‘KeyError’ exception
```

- Add Elements

```
>>>D['name'] = 'YHWH'

>>>D['age'] = 'The Ageless One'

>>>print D
```

- Test Membership

```
>>>'age' in D                    >>>'name' not in D
```

- Change Entry

```
>>>dict={'country':'India', 'state':'AP'}

>>>dict['state']='Andhra Pradesh'

>>>dict['city']='Hyderabad'                    # new entry
```

- Add Elements

```
>>>D['name'] = 'YHWH'

>>>D['age'] = 'The Ageless One'

>>>print D
```

- Test Membership

```
>>>'age' in D                    >>>'name' not in D
```

- Change Entry

```
>>>dict={'country':'India', 'state':'AP'}

>>>dict['state']='Andhra Pradesh'

>>>dict['city']='Hyderabad'                    # new entry
```

- Add Elements

```
>>>D['name'] = 'YHWH'

>>>D['age'] = 'The Ageless One'

>>>print D
```

- Test Membership

```
>>>'age' in D                    >>>'name' not in D
```

- Change Entry

```
>>>dict={'country':'India', 'state':'AP'}

>>>dict['state']='Andhra Pradesh'

>>>dict['city']='Hyderabad'                    # new entry
```

Dictionary Methods

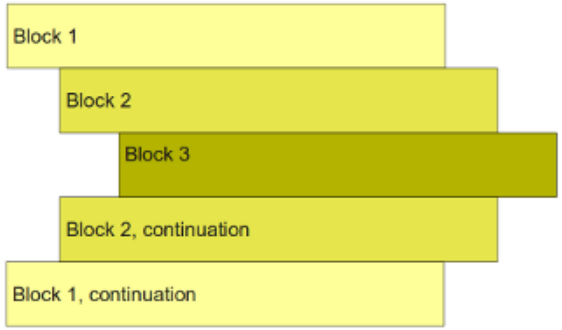
- `len(dict)`
- `dict.keys()`
- `dict.values()`
- `dict.items()`
- `dict.copy()`
- `dict.update(dict2)`

- `dict.pop(key)`
- `del dict[key]`
- `dict.clear()` `# empty dictionary`
- `del(dict)` `# entire dictionary`

How do you convert from lists to dictionaries and vice versa?

Flow Control Statements

- Structuring is through indentation. Welcome to Python :)



- New line signals end of statement; semicolon separates statements.
- Bracketed code can span multiple lines. Alternatively, use '\'.
- `True` ≡ any non-zero or non-empty objects.
- `False` ≡ zero, empty object, or `None`.

- Two-way

```
if <test>:                              # notice the colon
   <statements 1>                       # the indentation too
else:
   <statements 2>                       # optional else
```

- The Ternary Expression

  e.g. `max = x if (x > y) else y`

- Multiway `if...elif`

  - No `switch` statement in Python.

- Two-way

```
if <test>:                                  # notice the colon
   <statements 1>                           # the indentation too
else:
   <statements 2>                           # optional else
```

- The Ternary Expression

  e.g. `max = x if (x > y) else y`

- Multiway `if...elif`
  - No `switch` statement in Python.

- Two-way

```
if <test>:                                  # notice the colon
   <statements 1>                           # the indentation too
else:
   <statements 2>                           # optional else
```

- The Ternary Expression

  e.g. `max = x if (x > y) else y`

- Multiway `if...elif`
  - No `switch` statement in Python.

Examples

```
seq = ‘AGCAATTTCAGGT’
if ‘GAC’ in seq:
      print ‘Eureka!’ * 3
else:
      print ‘Bye’ * 2
```

Examples

```
richter = input("Magnitude of the earth quake:")
if richter < 4.0:
    print ‘Minor’
elif 4.0 < richter < 6.0:
    print ‘Moderate’
else:
    print ‘Major’
```

Examples

```
seq = ‘AGCAATTTCAGGT’
if ‘GAC’ in seq:
      print ‘Eureka!’ * 3
else:
      print ‘Bye’ * 2
```

Examples

```
richter = input("Magnitude of the earth quake:")
if richter < 4.0:
    print ‘Minor’
elif 4.0 < richter < 6.0:
    print ‘Moderate’
else:
    print ‘Major’
```

Flow Control: `for` Loops

```
for <target> in <sequence object>:
    <statements>
else:
    <statements>
```

- Be indented!
- `<sequence object>` can be strings, lists, tuples *&* dictionaries.
- The *optional* `else` block executed if the sequence is exhausted (loop exits without `break` statement).

```
>>>for i in [2,4,6,8,10]:
      print i

>>>for seq in 'AGTCATGGA':
      print 'Base:', seq
```

Flow Control: `for` Loops

```
for <target> in <sequence object>:
    <statements>
else:
    <statements>
```

- Be indented!
- `<sequence object>` can be strings, lists, tuples *&* dictionaries.
- The *optional* `else` block executed if the sequence is exhausted (loop exits without `break` statement).

```
>>>for i in [2,4,6,8,10]:
       print i

>>>for seq in 'AGTCATGGA':
       print 'Base:', seq
```

The Range() Function

- Generates lists containing arithmetic progressions.
- range(start, end, step)

```
>>>range(10)                          # start = 0, last = end-1

>>>range(2,20,2)

>>>range(50,0,-1)

>>>range(0,9,5)

>>>L=['BI', 'CSE', 'VLSI', 'PGSSP', 'CL']

>>>for i in range(len(L)):
       print L[i]
```

Examples

1. $\sum_{n=1}^{100} n$

2. $\prod_{i=1}^{n} \sqrt{i}$

```
# get n from the user
```

```
>>>sum=0
>>>for i in range(1,101):
...     sum+=i
>>>print sum
```

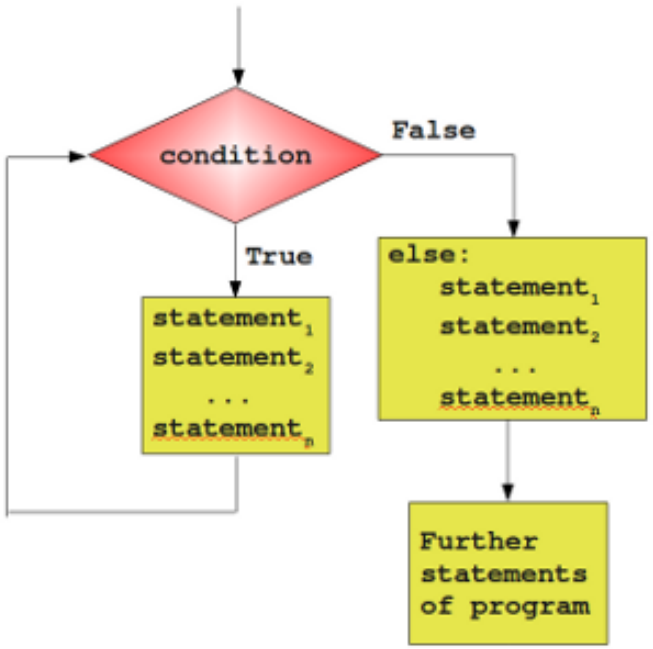
Examples

1. $\sum_{n=1}^{100} n$

2. $\prod_{i=1}^{n} \sqrt{i}$ `# get n from the user`

```
>>>sum=0
>>>for i in range(1,101):
...     sum+=i
>>>print sum
```

```
while <test>:
    <statements1>
else:
    <statements2>
```



Examples

$$\sum_{n=1}^{100} n$$

```
i=1
sum=0
while i <= 100:
    sum+=i
    i+=1
print sum
```

- The `break` and `continue` statements

Examples

$$\sum_{n=1}^{100} n$$

```
i=1
sum=0
while i <= 100:
    sum+=i
    i+=1
print sum
```

- The `break` and `continue` statements

Examples

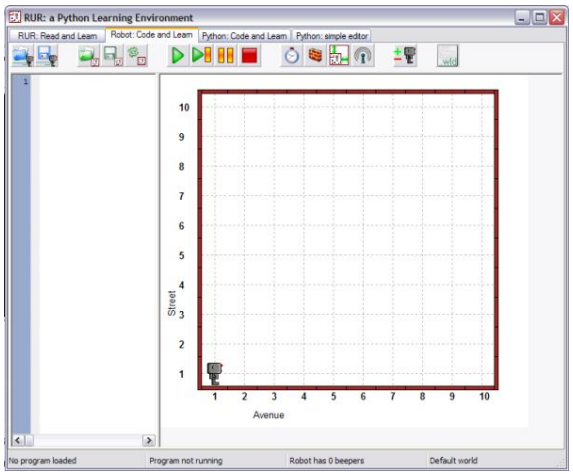
$$\sum_{n=1}^{100} n$$

```
i=1
sum=0
while i <= 100:
    sum+=i
    i+=1
print sum
```

- The `break` and `continue` statements

Fun with Python

- RUR-PLE - is an environment designed to help you learn computer programming using Python. Reeborg, a robot, will be at your service. You will be his master. Cool, huh? Make it do a task.



- The Python Challenge game, set of programming riddles solved using Python.