Argonne NATIONAL LABORATORY

Learning Python

Getting results for beamlines and scientific programming

4. Basic Python: some common statement types

The Advanced Photon Source is an Office of Science User Facility operated for the U.S. Department of Energy Office of Science by Argonne National Laboratory

U.S. DEPARTMENT OF ENERGY | Office of Science

Outline of topics to be covered

1. If statements
   - The pass placeholder
2. While loops
   - Use of break, continue and else in loops
3. For loops
   - range
   - zip

The Advanced Photon Source is an Office of Science User Facility operated for the U.S. Department of Energy Office of Science by Argonne National Laboratory

Tests: if statements

- The if statement has a test and a block of statements to be executed if the test is true.

```
if A > 0:
      print "positive"
      b = 'positive'
print 'test done'
```

  Note that Python uses indentation to tell when the block begins and ends. Python will accept any number of spaces/tabs as long as they are the same for the block – however always use 4 space indents, never use tabs; use a Python-aware editor that will make this easy.
- It is also possible to test if a variable is True or False

```
if A: print "true"
print 'test done'
```

  Note that indent is not needed for "one-line if"
  - For int and float, 0 is False, non-zero is true,
  - strings, lists, tuples, dicts: empty is false, otherwise true
  - None is false

The Advanced Photon Source is an Office of Science User Facility operated for the U.S. Department of Energy Office of Science by Argonne National Laboratory

Use of if with else and elif (*i.e.* "else if")

- If with else works as expected

```
if A > 0:
   print "positive"
   b = 'positive'
else:
   print "negative or zero"
print 'test done'
```

- Also possible to have multiple tests:

```
if A > 0:
   print "positive"
elif A < 0:
   print "negative"
else:
   print "zero"
print 'test done'
```

The Advanced Photon Source is an Office of Science User Facility operated for the U.S. Department of Energy Office of Science by Argonne National Laboratory

The pass statement

- A special statement, pass, is provided for when one needs a command as a placeholder (to manage indents):

```
if A > 0:
   print "positive"
elif A < 0:
   pass
else:
   print "zero"
print 'test done'
```


Indents nest and determine statement grouping

- The indent for the else clause determines which if statement pairs to it:

```
if A == 0:
   print "A zero"
   if B == 0:
      print "B zero"
else:
   print "A non-zero"
```

- Note difference with else indented

```
if A == 0:
   print "A zero"
   if B == 0:
      print "B zero"
   else:
      print "B non-zero"
```


Looping: while

- The while statement in Python is used to execute a block of statements until a test is no longer true

```
t = 0
while t < 10:
  t = t + 1
  print t
```

  This will print the numbers from 1 to 10
- Notes:
  - If the test fails on the first attempt, the loop contents are never run
  - Note that the Pythonic way to code "`t = t + 1`" is "`t += 1`"

Looping: break and continue

- The continue statement is used to bypass the rest of the code in a loop early

```
t = 0
while t < 10:
  t += 1
  if t == 2 or t == 3: continue
  print t
```

  - This prevents printing of 2 or 3 (printing 1,4,5,6,7,8,9,10)
- The break statement is used to exit the loop early

```
t = 0
while t < 10:
  t += 1
  if t == 5: break
  print t
print 'loop done'
```

  - This stops the loop when t is 5 (printing 1,2,3,4 and then "loop done")

Looping: the else statement

- The else statement in a loop is a bit weird at first. The contents of the else clause are executed only when the loop completes without a break statement. This is surprisingly useful once one gets used to having it.

```
t = 0
while t < s:
  t += 1
  if t == 10: break
  print t
else:
  print "s is <10"
```

  - This prints from 1 to s or 1 to 9, whichever is smaller. If it gets to 9, it also prints "s is < 10"

Looping: for

The for loop is very commonly used in Python because it can be used to execute a set of statements for every value in a tuple, list or dict (…). [Advanced topic: it works even better with generators]

- With a tuple:

```
for X in (2,"four",6.0):
  print X
```

  - This will sequentially assign each value in the tuple to variable X and thus print 2, four and 6.0 on successive lines

- Note that continue, break and else work exactly the same with for loops as with while loops

Using for loops with dicts

- If a dict is used as the source a for loop, the iteration variable is set to each key in the dict. Note that order of keys may change.

```
>>> d = {2:1,"four":2,6.0:3}
>>> for key in d:
...     print key
...
four
2
6.0
```

- If you want to control the sequence that keys are used, a sort (or some other ordering method) is needed. Now the order is fixed.
  - Note here we use values and keys

```
>>> for key in sorted(d.keys()):
...     print key,'=>',d[key]
...
2 => 1
6.0 => 3
four => 2
```


The range routine: plays well with for loops

- A very commonly used version of the for command is to use it with the range() built-in function to loop over a list of consecutive numbers

```
>>> for i in range(5):
...    print i
...
0
1
2
3
4
```

The range routine: actually quite versatile

The range routine can be used in a number of ways:

- range(n)
  - Generates a list of values: (0,1,…n-1) which will be empty if n<=0

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

- range(*start*,*stop*)
  - Generates a list of values: (*start*, *start*+1,…*stop*-1) which will be empty if *stop*<=*start*

```
>>> range(2,9)
[2, 3, 4, 5, 6, 7, 8]
>>> range(2,2)
[]
```

- range(start,stop,step)
  - Generates a list of values: (*start*, *start*+step,…*stop*-1)

```
>>> range(2,9,2)
[2, 4, 6, 8]
>>> range(2,-2,-1)
[2, 1, 0, -1]
```


Use of zip to loop over multiple items

- Sometimes one wants to loop over several lists at the same time, the zip routine is used to collate lists/tuples into a tuple of tuples:

```
>>> zip( ('a','b','c'), [11,22,33], range(5) )
[('a', 11, 0), ('b', 22, 1), ('c', 33, 2)]
```

  - Note that if the lists are not of equal length, the resulting tuple has the length of the shortest input item
- Using zip one can iterate over the lists (tuples) quite easily:

```
>>> for i,j,k in zip( ('a','b','c'), [11,22,33], range(5) ):
...     print 'i=',i,' j+k=',j+k
...
i= a  j+k= 11
i= b  j+k= 23
i= c  j+k= 35
```


Homework

- Using a `while` loop, write a short Python script to compute n factorial
  [=n*(n-1)*(n-2)*…2],
  where n is a Python variable containing a positive integer
- Do the same thing using a `for` loop with the `range` function