NPTEL MOOC

PROGRAMMING, DATA STRUCTURES AND ALGORITHMS IN PYTHON

Week 2, Lecture 4

Madhavan Mukund, Chennai Mathematical Institute
http://www.cmi.ac.in/~madhavan

A typical Python program

```
def function_1(..,..):
   …
def function_2(..,..):
   …
      ⋮
def function_k(..,..):
   …

statement_1
statement_2
   ⋮
statement_n
```

- Interpreter executes statements from top to bottom
- Function definitions are “digested” for future use
- Actual computation starts from `statement_1`

Control flow

- Need to vary computation steps as values change
- Control flow — determines order in which statements are executed
  - Conditional execution
  - Repeated execution — loops
  - Function definitions

Conditional execution

```
if m%n != 0:
   (m,n) = (n,m%n)
```

- Second statement is executed only if the condition `m%n != 0` is `True`
- Indentation demarcates body of `if` — must be uniform

```
if condition:
   statement_1  # Execute conditionally
   statement_2  # Execute conditionally
statement_3     # Execute unconditionally
```

Alternative execution

```
if m%n != 0:
    (m,n) = (n,m%n)
  else:
    gcd = n
```

* `else:` is optional

Shortcuts for conditions

- Numeric value `0` is treated as `False`
- Empty sequence `""`, `[]` is treated as `False`
- Everything else is `True`

```
if m%n:
   (m,n) = (n,m%n)
else:
   gcd = n
```

Multiway branching, `elif:`

```
if x == 1:
  y = f1(x)
else:
  if x == 2:
    y = f2(x)
  else:
    if x == 3:
      y = f3(x)
    else:
      y = f4(x)
```

```
if x == 1:
  y = f1(x)
elif x == 2:
  y = f2(x)
elif x == 3:
  y = f3(x)
else:
  y = f4(x)
```

Loops: repeated actions

- Repeat something a fixed number of times

```
for i in [1,2,3,4]:
  y = y*i
  z = z+1
```

- Again, indentation to mark body of loop

Repeating n times

- Often we want to do something exactly `n` times

```
for i in [1,2,..,n]:
    . . .
```

  - `range(0,n)` generates sequence `0,1,…,n-1`

```
for i in range(0,n):
    . . .
```

  - `range(i,j)` generates sequence `i,i+1,…,j-1`

    - More details about `range()` later

Example

- Find all factors of a number `n`
- Factors must lie between `1` and `n`

```
def factors(n):
    flist = []
    for i in range(1,n+1):
        if n%i == 0:
            flist = flist + [i]
    return(flist)
```

Loop based on a condition

- Often we don't know number of repetitions in advance

```
while condition:
    . . .
```

- Execute body if `condition` evaluates to `True`
- After each iteration, check `condition` again
- Body must ensure progress towards termination!

Example

- Euclid's gcd algorithm using remainder
- Update `m`, `n` till we find `n` to be a divisor of `m`

```
def gcd(m,n):
   if m < n:
      (m,n) = (n,m)
   while m%n != 0:
      (m,n) = (n,m%n)
   return(n)
```

Summary

- Normally, statements are executed top to bottom, in sequence
- Can alter the control flow
  - `if … elif … else` — conditional execution
  - `for i in …` — repeat a fixed number of times
  - `while …` — repeat based on a condition