

NPTEL MOOC

PROGRAMMING, DATA STRUCTURES AND ALGORITHMS IN PYTHON

Week 2, Lecture 1

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`

A more messy program

```
statement_1
```

```
def function_1(..., ...):  
    ...
```

```
statement_2
```

```
statement_3
```

```
def function_2(..., ...):  
    ...
```

```
statement_4
```

```
⋮
```

- * Python allows free mixing of function definitions and statements
- * But programs written like this are likely to be harder to understand and debug

Assignment statement

- * Assign a **value** to a **name**

`i = 5`

`j = 2*i`

`j = j + 5`

- * Left hand side is a **name**
- * Right hand side is an **expression**
 - * Operations in expression depend on **type** of value

Numeric values

- * Numbers come in two flavours
 - * `int` — integers
 - * `float` — fractional numbers
- * `178, -3, 4283829` are values of type `int`
- * `37.82, -0.01, 28.7998` are values of type `float`

int vs float

- * Why are these different types?
- * Internally, a value is stored as a finite sequence of 0's and 1's (binary digits, or bits)
- * For an `int`, this sequence is read off as a binary number
- * For a `float`, this sequence breaks up into a **mantissa** and **exponent**
 - * Like “scientific” notation: 0.602×10^{24}

Operations on numbers

- * Normal arithmetic operations: $+$, $-$, $*$, $/$
 - * Note that $/$ always produces a float
 - * $7/3.5$ is 2.0 , $7/2$ is 3.5
- * Quotient and remainder: $//$ and $\%$
 - * $9//5$ is 1 , $9\%5$ is 4
- * Exponentiation: $**$
 - * $3**4$ is 81

Other operations on numbers

- * `log()`, `sqrt()`, `sin()`, ...
- * Built in to Python, but not available by default
- * Must include `math` “library”
 - * `from math import *`

Names, values and types

- * Values have types
 - * Type determines what operations are legal
- * Names inherit their type from their current value
 - * Type of a name is not fixed
- * Unlike languages like C, C++, Java where each name is “declared” in advance with its type

Names, values and types

- * Names can be assigned values of different types as the program evolves

```
i = 5      # i is int
```

```
i = 7*1    # i is still int
```

```
j = i/3    # j is float, / creates float
```

```
...
```

```
i = 2*j    # i is now float
```

- * `type(e)` returns type of expression `e`
- * Not good style to assign values of mixed types to same name!

Boolean values: `bool`

- * `True`, `False`
- * Logical operators: `not`, `and`, `or`
 - * `not True` is `False`, `not False` is `True`
 - * `x and y` is `True` if both of `x,y` are `True`
 - * `x or y` is `True` if at least one of `x,y` is `True`

Comparisons

- * $x == y$, $a != b$,
 $z < 17 * 5$, $n > m$,
 $i \leq j + k$, $19 \geq 44 * d$
- * Combine using logical operators
 - * $n > 0$ and $m \% n == 0$
- * Assign a boolean expression to a name
 - * $\text{divisor} = (m \% n == 0)$

Examples

```
def divides(m,n):  
    if n%m == 0:  
        return(True)  
    else:  
        return(False)
```

```
def even(n):  
    return(divides(2,n))
```

```
def odd(n):  
    return(not divides(2,n))
```

Summary

- * Values have types
 - * Determine what operations are allowed
- * Names inherit type from currently assigned value
 - * Can assign values of different types to a name
- * `int, float, bool`