



MIT Global Startup Labs

<http://aiti.mit.edu>

Indonesia Summer 2013
Meetup 07 – Introduction to Python



Today's Meetup

- Why Python?
- Basic Syntax
- Variables
- Control Statements
- Functions

Why Python?

Python because...

- Convenient built-in functions and data structures
- Great for rapid prototyping
 - No separate compile step
 - No need to explicitly specify method argument types beforehand
- Syntax is readable and fast to write

The diagram illustrates the transformation of C-style nested if statements into Python-style nested if statements. On the left, a light blue box contains the following C-style code:

```
if (x)
{
    if (y)
    {
        a();
    }
    b();
}
```

An arrow points from this code to a second light blue box on the right, which contains the equivalent Python-style code:

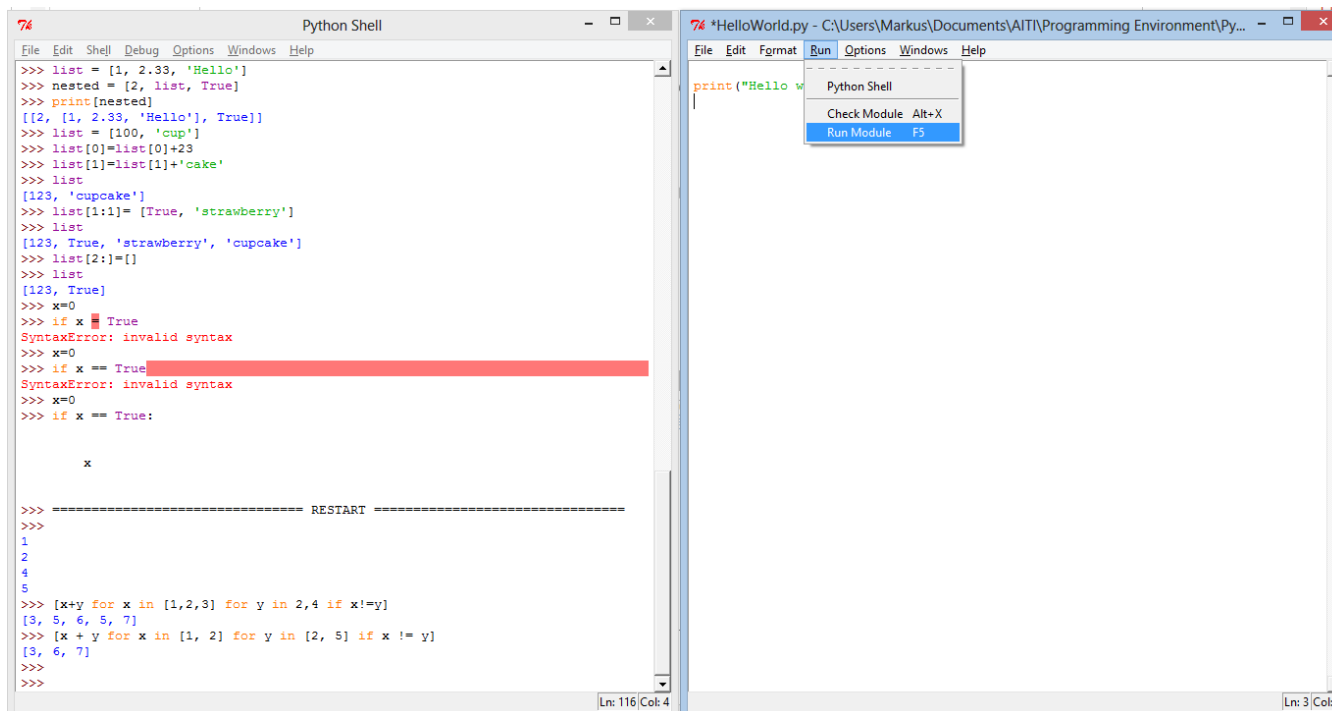
```
if x:
    if y:
        a()
    b()
```

Python because...

- We want each of you to reach millions of users, and don't want to waste time building the pipes and plumbing
- Python is supported by a number of good frameworks, including
 - Django
 - Heroku
 - Google AppEngine

Python Shell and IDLE

- Download Python 2.7.5:
<http://www.python.org/getit/>



The image shows two windows from the Python 2.7.5 environment. The left window is the Python Shell, displaying a series of commands and their outputs. The right window is the IDLE editor, showing a file named 'HelloWorld.py' with a single line of code: `print("Hello w...")`. A context menu is open over the IDLE window, showing options: 'Python Shell', 'Check Module Alt+X', and 'Run Module F5'.

```
Python Shell
File Edit Shell Debug Options Windows Help
>>> list = [1, 2.33, 'Hello']
>>> nested = [2, list, True]
>>> print[nested]
[[2, [1, 2.33, 'Hello'], True]]
>>> list = [100, 'cup']
>>> list[0]=list[0]+23
>>> list[1]=list[1]+'cake'
>>> list
[123, 'cupcake']
>>> list[1:1]= [True, 'strawberry']
>>> list
[123, True, 'strawberry', 'cupcake']
>>> list[2:]=[]
>>> list
[123, True]
>>> x=0
>>> if x True
SyntaxError: invalid syntax
>>> x=0
>>> if x == True
SyntaxError: invalid syntax
>>> x=0
>>> if x == True:
    x

----- RESTART -----
>>>
1
2
4
5
>>> [x+y for x in [1,2,3] for y in 2,4 if x!=y]
[3, 5, 6, 5, 7]
>>> [x + y for x in [1, 2] for y in [2, 5] if x != y]
[3, 6, 7]
>>>
>>>
```

```
*HelloWorld.py - C:\Users\Markus\Documents\AITI\Programming Environment\Py...
File Edit Format Run Options Windows Help
print("Hello w...
Python Shell
Check Module Alt+X
Run Module F5
Ln: 3 Col: 0
```

Basic Syntax

Basic Syntax

- Semicolons are only used to separate multiple statements on the same line (which is discouraged)

- Whitespace is important!

```
if x:
    if y:
        a()
    b()
```

- Use hash # to write comments

```
>>> # this is a comment
```


Variables

Variables

Python is a “dynamically typed” language

- A variable’s data type is not declared.
- “Statically typed” languages like Java must declare a variable’s data type: `String x = “Hello World”`

- ⇒ Python automatically detects which type your variable is
- ⇒ Use `type()` function to get variable’s type

```
>>> x = 5
>>> type(x)
<type 'int'>
>>> y = 12.43
>>> type(y)
<type 'float'>
>>> z = True
>>> type(z)
<type 'bool'>
>>> a = 'Hello'
>>> type(a)
<type 'str'>
>>> |
```

Numbers

- Can assign multiple values: $x = y = 0$
- Can use basic operators $+$, $-$, $*$, $/$, $\%$, $**$
- Complex numbers are supported
 - $a = 1 + 5j$
 - `Complex(1,5)`
 - Use `a.real`, `a.imag`, `abs(a)` to get the real part, imaginary part or the absolute value of the complex number

Number example

```
>>>x = y = 3
```

```
>>>z = 14
```

```
>>>z/x
```

```
>>>z%y**2
```

Strings

- Use single or double quotes to declare strings
- Use `\n` to start a new line, `\` to continue in same line
- Use triple quotes for multiple line strings

```
>>> print "In this example I start \n\  
a new line and \n\  
    use whitespace in \  
    one string"  
In this example I start  
a new line and  
    use whitespace in         one string  
>>> |
```

```
>>> print """ Alternatively we can  
just use triple quotes  
    to do all of that"""  
Alternatively we can  
just use triple quotes  
    to do all of that  
>>>
```

String operators

- We can add or multiply strings
- We can find the length of a string by using `len()`
- Use `_` to refer to the last value used

```
>>> a = 'yee'  
>>> b = 'haah'  
>>> a+b  
'yeehaah'  
>>> 3*a + b + '!!!'  
'yeeyeeehaah!!!'  
>>> len(_)  
16  
>>>
```

Strings as a sequence

- We can access single letters of a string by using []
- BUT: We cannot change single characters of a string

```
>>> word = 'Hello World'
>>> word[0]
'H'
>>> word[3:6]
'lo '
>>> word[6:]
'World'
>>> word[-3]
'r'
>>>
```

Strings example

```
>>> x = Hello
```

```
>>> x[:2] + x[2:]
```

```
>>> x + x[-1]*2
```

```
>>> x[1] = ,a'
```

```
>>> print(x)
```


Lists

- A list is a sequence of values
- The elements do not have to be of the same type: can be ints, floats, strings or even other lists mixed together
- We can access and change single elements using []

```
>>> list = [1, 2.33, 'Hello']
>>> nested = [2, list, True]
>>> print(nested)
[[2, [1, 2.33, 'Hello'], True]]
>>> |
```

Operating on lists

- Access single elements and change them
- We can slice lists, add items to lists or delete items from list

```
>>> list = [100, 'cup']
>>> list[0]=list[0]+23
>>> list[1]=list[1]+'cake'
>>> list
[123, 'cupcake']
>>> list[1:1]= [True, 'strawberry']
>>> list
[123, True, 'strawberry', 'cupcake']
>>> list[2:]=[]
>>> list
[123, True]
>>>
```

Using methods with lists

- `list.append(x)`
 - Add an item to the end of the list; equivalent to `a[len(a):] = [x]`.
- `list.insert(i, x)`
 - Insert an item `x` at a given position `i`
- `list.remove(x)`
 - Remove the first item from the list whose value is `x`
- `list.pop([i])`
 - Remove the item at the given position in the list, and return it.
- `list.count(x)`
 - Return the number of times `x` appears in the list.
- `list.sort()`
 - Sort the items of the list, in place.
- `list.reverse()`
 - Reverse the elements of the list, in place.
- And more

List comprehension

- A fast way to create lists that follow a specified pattern
- Use for loops and if statements to control list content

What does
this code
generate?

```
squares = []  
for x in range(10):  
    squares.append(x**2)  
  
squares = [x**2 for x in range(10)]
```

List comprehension

We can also use more than one sequence to generate a list:

What does the following statement generate?

```
>>> [x + y for x in [1, 2] for y in [2, 5] if x != y]
```

User input

- We can ask the user for an input and store the value in a variable:
- Use *raw_input* to get string values
>>> name = raw_input(,What is your name?')
- Use *input* to get number values
>>> age = input(,How old are you?')

Control Statements

Control statements

- Conditionals: control which set of statements is executed.
 - if / else
- Iteration: control how many times a set of statements is executed.
 - while loops
 - for loops

If statements

Consist of a condition and a body:

- If the condition is true then the body gets executed
- Conditions can use <, >, ==, !=, &&, ...
- The body can be any python operation
- Indentation is key!

```
if condition1:  
    body1  
elif condition2:  
    body2  
else:  
    body3
```

If example

```
>>> x=12
```

```
>>> if x=0:
```

```
    print(„x is zero“)
```

```
elif x>0:
```

```
    print(„x is greater than zero“)
```

```
else:
```

```
    print(„x is less than zero“)
```

```
print(„Indentation is key“)
```

While loops

- As long as the condition is true, the body gets executed repeatedly

– Loop ends as soon as condition turns false

We can use break and continue statements to break the cycle:

- Break: breaks out of loop completely
- Continue: skips the rest of the body for one cycle, but continues loop

```
while condition:  
    body
```

While loop example

```
>>> i = 0
```

```
>>> while i < 5:
```

```
    i = i + 1
```

```
    if i == 3
```

```
        continue
```

```
    print i
```

For loops

- Execute a body over a sequence of values
- We can use break and continue statements in for loops as well
- Sequences can be lists, strings, or generated by the range() function

```
for element in sequence:  
    body  
  
for i in [0,1,2,3]:  
    print i
```

The range() function

- Use range() to create sequences:
 - range(4) -> [0, 1, 2, 3]
 - range(5, 10) -> [5, 6, 7, 8, 9]
 - range (1, 30, 5) -> [1, 6, 11, 16, 21, 26]

```
for i in [0,1,2,3]:  
    print i  
  
for i in range(4):  
    print i
```

For loop example

```
for i in range(2, 10):  
    if i%2 == 0:  
        print i, "is even"  
        continue  
    print i, "is odd"
```

Functions

Functions

- A function is a sequence of python statements that operates on predefined input parameters and can be called by a prespecified name

Function definition

```
def functionname(parameter1, parameter2):  
    body
```

Calling the function

```
functionname(100, 'Hello')
```

Today's Assignment

Lab 4: Python Introduction

- 1. Fibonacci
- 2. Zeller's Algorithm
- 3. Rock Paper Scissors

Submit your solutions to the box, happy coding

