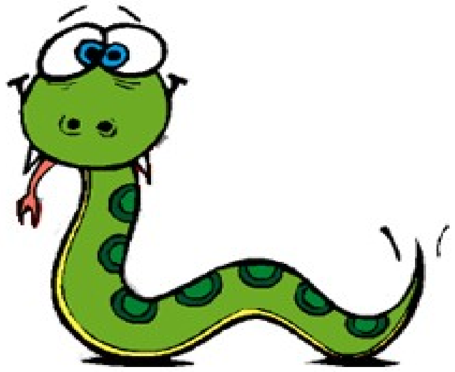
Pengantar python

www.python.org



Introduction

What is Python ?

- Compromise between shell script and C++/Java program
- Intuitive syntax
- Interpreted (sort of)
- Dynamically typed
- High-level datatypes
- Module system
- Just plain awesome

Introduction

Java

```
public class Hello {
      public static void main (String[] args)
{
            System.out.println("Hello, world!");
      }
}
```

Introduction

C++

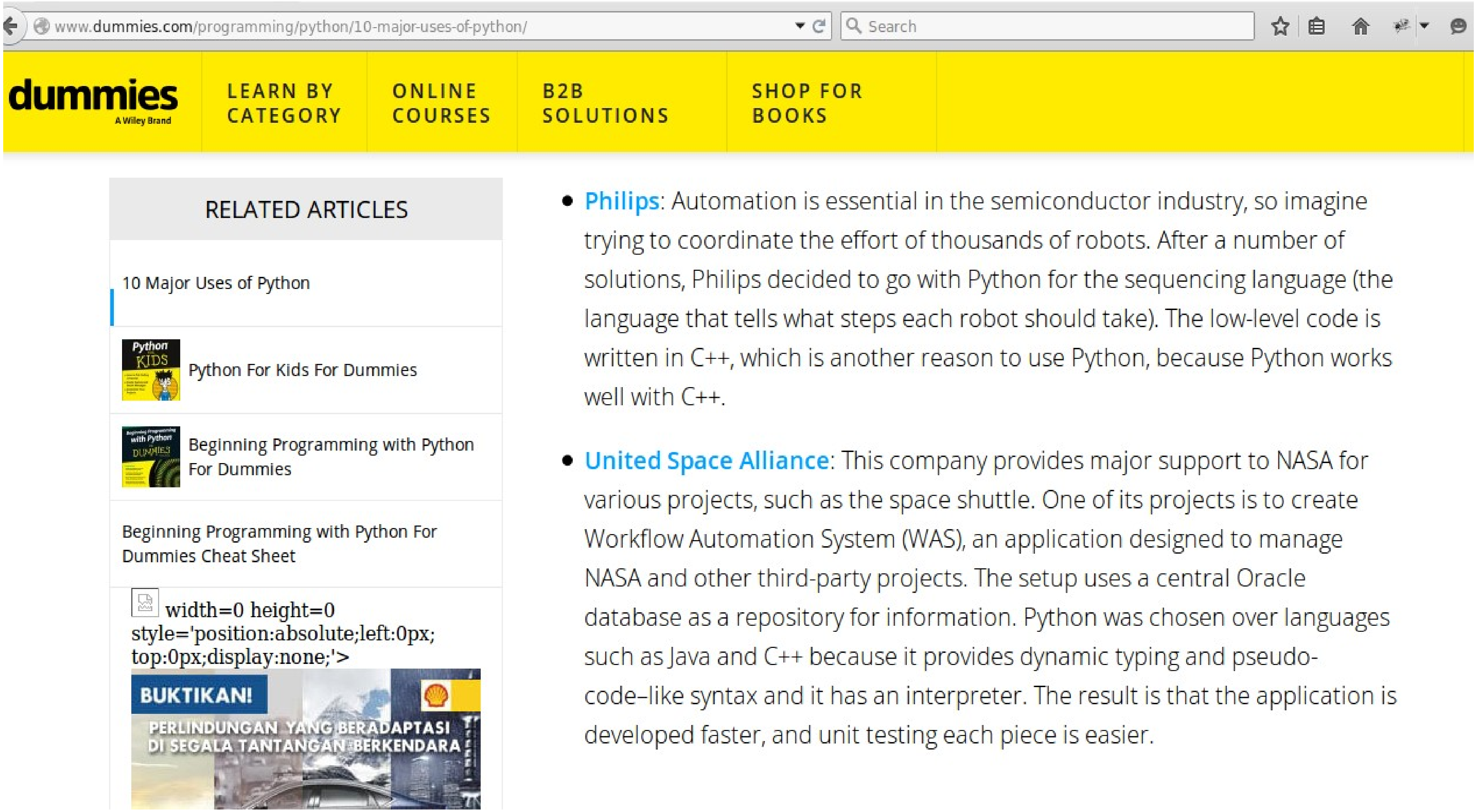
```
#include <iostream>
int main()
{
	std::cout << "Hello World!" << std::endl;
	return 0;
}
```

Introduction

Python

```
print(“Hello World”)
```

Python success story



www.dummies.com/programming/python/10-major-uses-of-python/

Search

dummies A Wiley Brand

LEARN BY CATEGORY | ONLINE COURSES | B2B SOLUTIONS | SHOP FOR BOOKS

RELATED ARTICLES

10 Major Uses of Python

Python For Kids For Dummies

Beginning Programming with Python For Dummies

Beginning Programming with Python For Dummies Cheat Sheet

width=0 height=0 style='position:absolute;left:0px; top:0px;display:none;'>

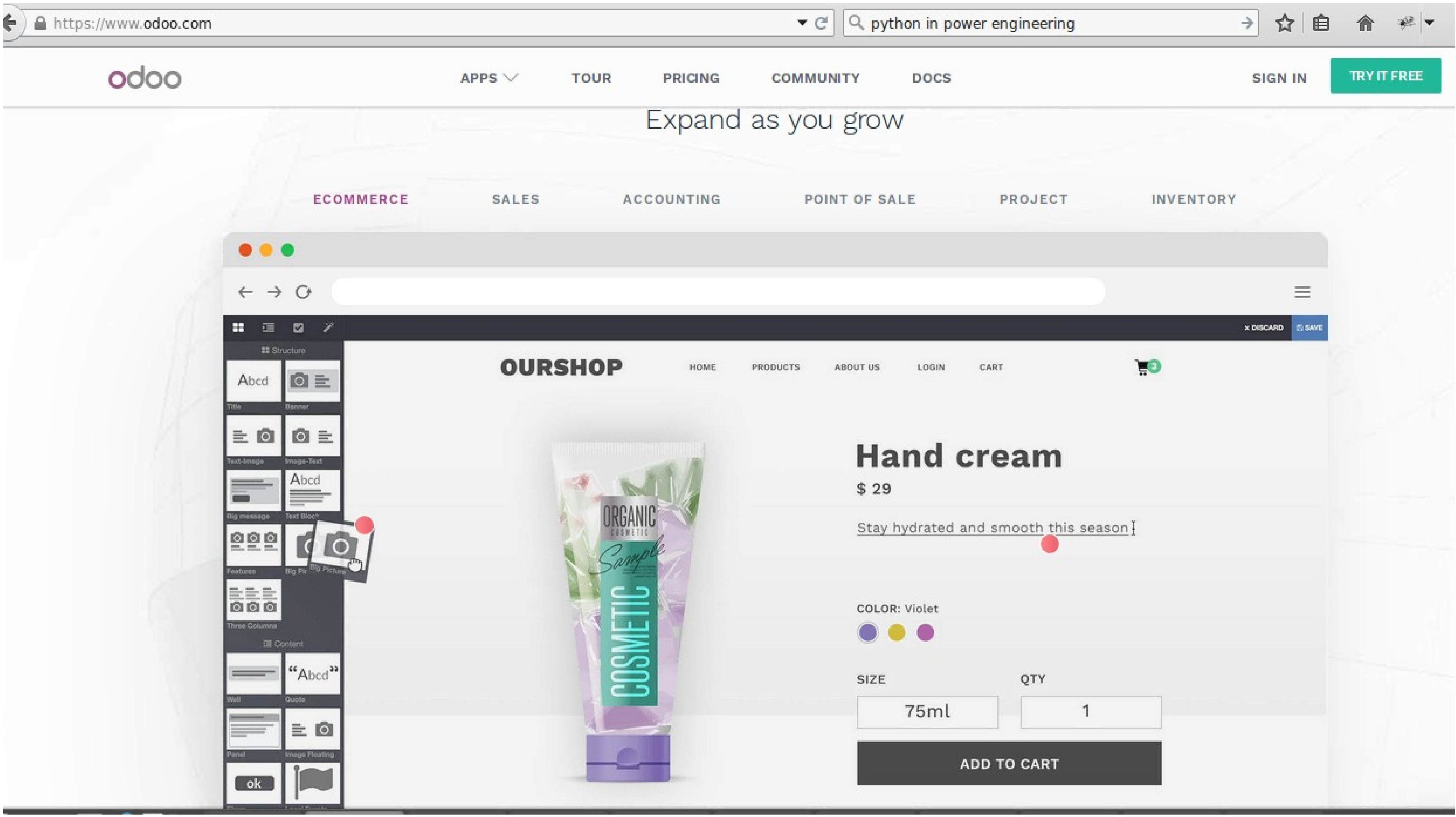
BUKTIKAN!

PERLINDUNGAN YANG BERADAPTASI DI SEGALA TANTANGAN BERKENDARA

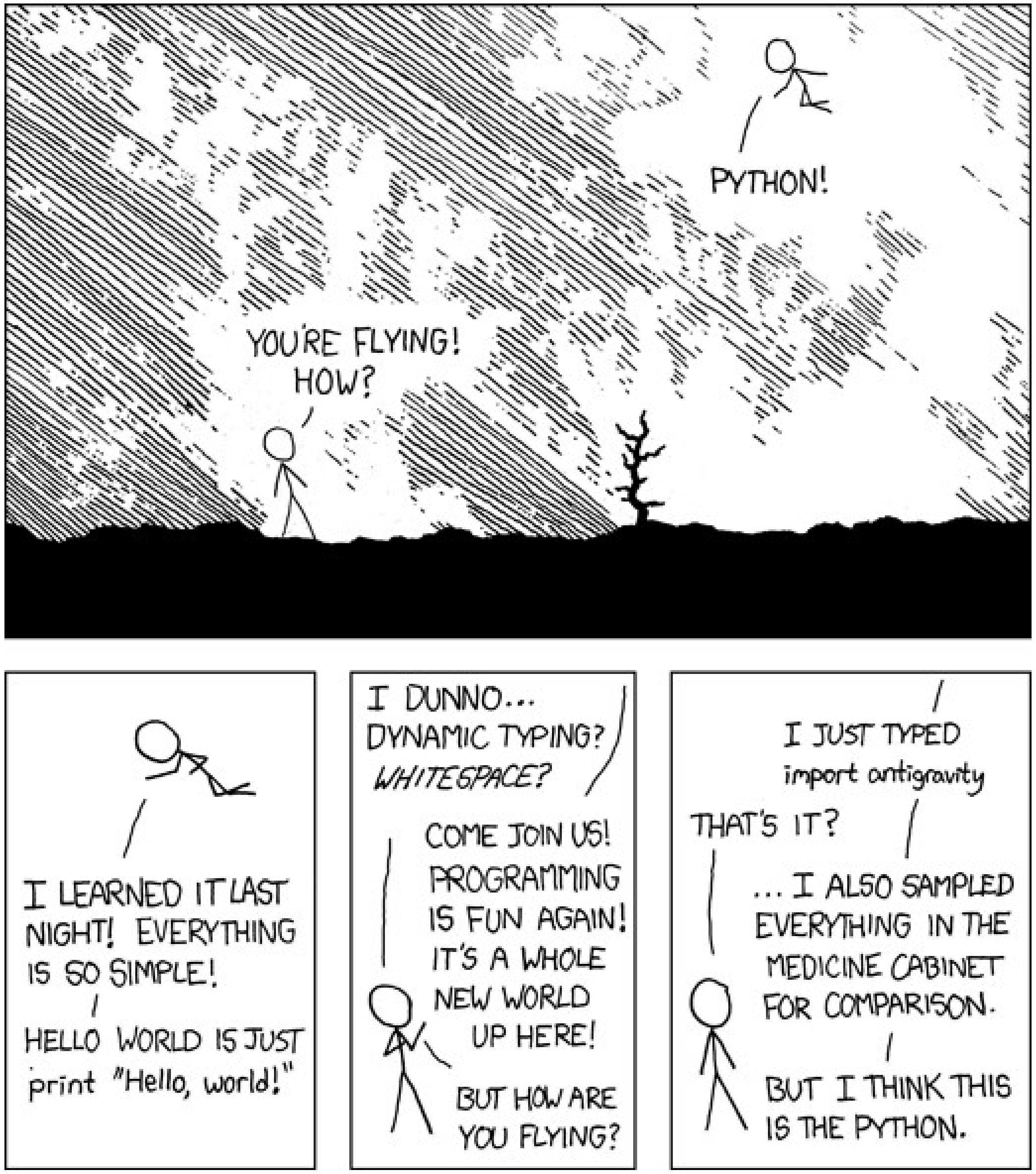
- Philips: Automation is essential in the semiconductor industry, so imagine trying to coordinate the effort of thousands of robots. After a number of solutions, Philips decided to go with Python for the sequencing language (the language that tells what steps each robot should take). The low-level code is written in C++, which is another reason to use Python, because Python works well with C++.

- United Space Alliance: This company provides major support to NASA for various projects, such as the space shuttle. One of its projects is to create Workflow Automation System (WAS), an application designed to manage NASA and other third-party projects. The setup uses a central Oracle database as a repository for information. Python was chosen over languages such as Java and C++ because it provides dynamic typing and pseudo-code–like syntax and it has an interpreter. The result is that the application is developed faster, and unit testing each piece is easier.

Python success story



Introduction


PYTHON!
YOU'RE FLYING! HOW?
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

Python

- What does it mean for a language to be “interpreted?”
- Trick question – “interpreted” and “compiled” refer to implementations, not languages
- The most common Python implementation (CPython) is a mix of both
    - Compiles source code to byte code (.pyc files)
    - Then interprets the byte code directly, executing as it goes
    - No need to compile to machine language
    - Essentially, source code can be run directly

Python

How do you use it?

- Write code interactively in the interpreter

```
Python 2.7.6 (default, Mar 22 2014, 22:59:56)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- Run a file in the interpreter with import file
- Run a file on the command line with python file.py

Basics

```
>>> 1 + 1
2
>>> print("hello world")
hello world
>>> x = 1
>>> y = 2
>>> x + y
3
>>> print (x)
1
```

Types

What does “dynamically typed” mean?

Types

What does “dynamically typed” mean?

- Variable types are not declared
- Python figures the types out at runtime

Types

type function:

```
>>> type(x)
<type 'int'>
```

isinstance function:

```
>>> isinstance(x, int)
True
```

Types

```
>>> x = 3
>>> x = "hello"
```

- Has x changed type?
- No – x is a name that points to an object
- First we make an integer object with the value 3 and bind the
- name ’x’ to it
- Then we make a string object with the value hello, and rebind
- the name ’x’ to it
- Objects do not change type

Types

Interpreter keeps track of all types and doesn’t allow you to do

things that are incompatible with that type:

```
>>> "hi" + 5
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
```

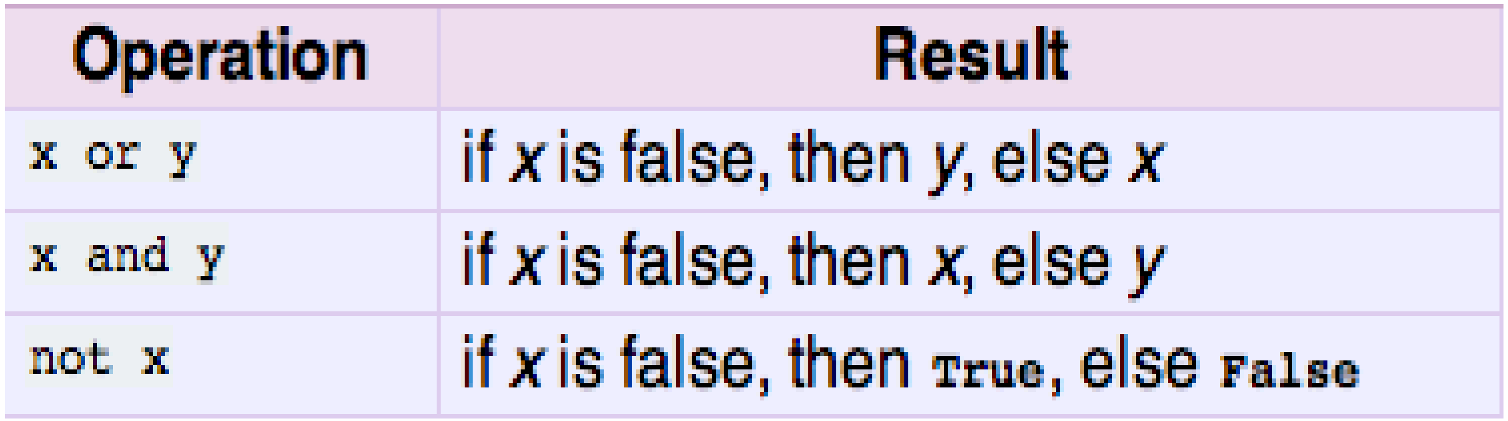
Datatypes

- None
- Booleans (True, False)
- Integers, Floats
- Sequences
  - Lists
  - Tuples
  - Strings
  - Dictionaries
- Classes and class instances
- Modules and packages

Booleans

- Booleans: True, False
- The following act like False:
  - None
  - 0
  - Empty sequences
- Everything else acts like True

Booleans: Operations



Operation	Result
`x or y`	if *x* is false, then *y*, else *x*
`x and y`	if *x* is false, then *x*, else *y*
`not x`	if *x* is false, then `True`, else `False`

- and, or both return one of their operands
- and, or are short-circuit operators

Booleans: Examples

```
>>> (2 + 4) or False
6
>>> not True
False
>>> not 0
True
>>> 0 and 2
0
>>> True and 7
7
```

Integers and Floats

- Numeric operators: + - * / % **
- No i++ or ++i, but we do have += and -=

Ints vs. Floats

```
>>> 5/2
2
>>> 5/2.
2.5
>>> float(5)/2
2.5
>>> int(5.2)
5
```

Assignments

```
>>> a = b = 0
>>> a, b = 3, 5
```

Something cool:

```
>>> a, b = b, a
>>> a
5
>>> b
3
```

Comparisons

```
>>> 5 == 5
True
>>> "hello" == "hello"
True
>>> 1 != 2
True
>>> 5 > 3
True
>>> "b" > "a"
True
```

Type casting

```
>>> a = 2000
>>> pesan = “Harga rata rata adalah Rp. “
>>> print(pesan + a) ←---Error
>>> print(pesan + str(a) ) ←-- Good

>>> x = 2
>>> y = 45
>>> z = y / x
>>> r = float(y) / x
>>> s = float(y) / float (x)
>>> b = “2”
>>> c = 6
>>> e = b + c ←-- Error
>>> f = int(b) + c ←--Good
```

If Statements

```
if a == 0:
    print ("a is 0")
elif a == 1:
    print ("a is 1")
else:
    print ("a is something else")
```

If Statements

- Don’t need the elif or else
- Condition can be any value, not just Boolean

```
if 5:
    print ("hello")
if "hello":
    print (5)
```

For Loops

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

Ranges

- range(n) produces [0, 1, ..., n-1]
- range(i, j) produces [i, i+1, ..., j-1]
- range(i, j, k) produces [i, i+k, ..., m]

```
>>> range(5, 25, 3)
[5, 8, 11, 14, 17, 20, 23]
```

Break and Continue

```
>>> for i in range(5):
...       print(i)
...       if i < 3:
...           continue
...       break
...
0
1
2
3
```

While Loops

```
>>> i = 0
>>> while i <= 3:
...         print(i)
...         i += 1
...
0
1
2
3
```

raw_input() dan input()

```
>>> nama = raw_input(“Masukkan nama Anda: “)
>>> type(nama)
>>> print(nama)

>>> nilai = input(“Masukkan nilai: “)
>>>type(nilai)
>>>print (nilai)
```

sys.argv

- Argument system sebagai input

```
import sys
arg0 = sys.argv[0]
arg1 = sys.argv[1]
print (“Argument 0 :” + arg0)
print (“Argument 1 :” + arg1)
```

Function

- Built in function :
  - max(), raw_input(), str() etc.
- User defined function:

```
def cetak(data):
   print(data)

def tambah(x,y):
  r = x + y
  return r
```

- Call function:

```
cetak("hwllo world")
print(tambah(2,4))
```

class

- **Class**: A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.
- **Data member**: A class variable or instance variable that holds data associated with a class and its objects.
- **Method** : A special kind of function that is defined in a class definition.
- **Object:** A unique instance of a data structure that's defined by its class. An object comprises both data members (class variables and instance variables) and methods.