Syntax

Hello World

```
print("hello world")
```

Indentation

- Most languages don’t care about indentation.
- Most humans do.
- We tend to group similar things together.

Indentation

```
//java
if(true)
        if(false)
                System.out.println("Hello1 !");
else
System.out.println("Hello2 !");
```

- The else here actually belongs to the 2nd if statement.

Indentation

```
//java
if(true){
        if(false){
                System.out.println("Hello1 !");
}else{
System.out.println("Hello2 !");
}}
```

- The else here actually belongs to the 2nd if statement.

Indentation

```
//java
if(true)
if(false)
System.out.println("Hello1 !");
else
System.out.println("Hello2 !");
```

- I knew a coder like this :D

Indentation

```
//java
if(true){
        if(false){
                System.out.println("Hello1 !");
        }else{
                System.out.println("Hello2 !");
        }
}
```

- You should always be explicit.

Indentation

```
# python
if True:
    if False:
        print("Hello1 !")
    else:
        print("Hello2 !")
```

- Python embraces indentation.

Comments

- (#) A traditional one line comment
- “””
  any string not assigned to a variable is considered comment.
  this is an example of a multi-line comment.
  “””
- “ this is a single line comment ”

Types

strings

```
# This is a string
Name = "Ahmad Hussein (that's me) "
# This is also a string
Home = 'Damascus, Syria’
# This is a multi-line string
Sites = ''' You can find me online on
    Sites like LinkedIn and Facebook.'''
# This is also a multi-line string
Bio = """ If you don’t find me online
    You can find me outside. """
```

Numbers

```
# Integer Number
year = 2010
year = int("2010")
# Floating Point Numbers
pi = 3.14159265
pi = float("3.14159265")
# Fixed Point Number
from decimal import Decimal
price = Decimal("0.02")
```

Null

Optional_data = None

Lists

```
# Lists can be heterogenous
favorites = []
# Appending
favorites.append(42)
# Extending
favorites.extend(['Python' , True])
# Equivalent to
favorites = [42 , 'Python' , True]
```

Lists

```
numbers = [ 1 , 2 , 3 , 4 , 5 ]
len(numbers)
# 5
numbers[0]
# 1
numbers[0:2]
# [1 ,2]
numbers[2:]
# [3, 4, 5]
```

Dictionaries

```
person = {}
# Set by key / Get by key
person['name'] = 'Ahmad Hussein'
# Update
person.update({
'favorites' : [42 , 'food'] ,
'gender' : 'male'
})
# Any immutable object can be a dictionary key
person[42] = 'favorite number'
person[(44.47 , -73.21)] = 'coordinates'
```

Dictionary Methods

```
person = { 'name' : 'karim' , 'gender' : 'Male'}
person['name']
person.get('name')
# 'karim'
person.keys()
# ['name' , 'gender']
person.values()
# ['karim' , 'Male']
person.items()
# [['name' , 'karim'] , ['gender' , 'Male']]
```

Booleans

Is_python = True

 # Everything in python can be cast to Boolean

Is_pyhton = bool(*'any object')*

 # All of these things are equivalent to False

These_are_false = False or 0 or *"" or {} or [] or None*

 # Most everything else is equivalent to True

These_are_true = True and 1 and *'Text' and {'a' : 'b'} and ['c','d']*

Operators

Arithmetic

```
a = 10      # 10
a += 1      # 11
a -= 1      # 10
b = a + 1   # 11
c = a - 1   # 9
d = a * 2   # 20
e = a / 2   # 5
f = a % 3   # 1
g = a ** 2  # 100
```

String Manipulation

```
animals = 'Cats ' + 'Dogs '
animals +=  'Rabbits'
# Cats Dogs Rabbits
fruit = ', '.join(['Apple' , 'Banana' , 'Orange'])
#Apple, Banana, Orange
date = '%s %d %d' % ('Mar' , 20 , 2018)
# Mar 20 2018
name = '%(first)s %(last)s' % {
'first' : 'Ahmad' ,
'last' : 'Hussein'}
# Ahmad Hussein
```

Logical Comparison

 # Logical And

a and b

 # Logical Or

a or b

 # Logical Negation

not a

 # Compound

(a and not (b or c))

Identity Comparison

1 is 1 == True

 # Non Identity

1 is not *'1' == True*

 # Example

bool(1) == True

bool(True) == True

1 and True == True

1 is True == False

Arithmetic Comparison

 # Ordering

a > b

a >= b

a < b

a <= b

 # Equality / Difference

a == b

a != b

Control Flow

Conditionals

```
grade = 82
if grade >= 90:
    if grade == 100:
        print('A+')
    else:
        print ('A')
elif grade >= 80:
    print ('B')
elif grade >= 70:
    print ('C')
else:
    print('E')
# B
```

For Loop

```
for x in range(10):
    print(x)

# 0-9

fruits = ['Apple' , 'Orange']
for fruit in fruits:
    print(fruit)
```

Expanded For Loop

```
countries = {
'SY' : 'Syria' ,
'US' : 'United States'
}
for key, value in countries.items():
    print('%s : %s' % (key, value))
```

While Loop

```
x = 0
while x < 100 :
    print(x)
    x +=1
```

List comprehensions

- Useful for replacing simple for-loops.

```
odds = [x for x in range(50) if x % 2]
```

```
odds = []
for x in range(50):
    if x % 2 :
        odds.append(x)
```

functions

```
def my_function():
    """Function Documentation """
    print('Hello Word')

def add( x , y ):
    return x + y
```

- Assignment : Fibonacci method !

Classes

Class Declaration

- The simplest class possible is shown in the following example:

```
class Person:
    pass  # An empty block
```

Class Methods

- **The self:**

  Class methods have only one specific difference from ordinary functions. they must have an extra first name that has to be added to the beginning of the parameter list, but you **do not** give a value for this parameter when you call the method, Python will provide it. This particular variable refers to the object *itself*, and by convention, it is given the name self.

The self in Python is equivalent to the this pointer in C++ and the this reference in Java and C#.

Class Methods

```
class Person:
    def say_hi(self):
        print('Hello, how are you?')

p = Person()
p.say_hi()

# Hello, how are you?
```

The __init__ method

- The __init__ method is run as soon as an object of a class is created. The method is useful to do any initialization you want to do with your object.

```
class Person:
    def __init__(self, name):
        self.name = name

    def say_hi(self):
        print('Hello, my name is', self.name)

p = Person('Yasser')
p.say_hi()
# Hello, my name is Yasser
```

Class And Object Variables

- **Class variables** are shared - they can be accessed by all instances of that class.

- **Object variables** are owned by each individual object/instance of the class. In this case, each object has its own copy of the field i.e. they are not shared and are not related in any way to the field by the same name in a different instance.

Class And Object Variables

```
class Robot:
    """Represents a robot, with a name."""
    # A class variable, counting the number of robots
    population = 0
    def __init__(self, name):
        """Initializes the data."""
        self.name = name
        print("(Initializing {})".format(self.name))
        # When this person is created, the robot
        # adds to the population
        Robot.population += 1
```

Class And Object Variables

```
def die(self):
    """I am dying."""
    print("{} is being destroyed!".format(self.name))
    Robot.population -= 1

def say_hi(self):
    print("Greetings, my masters call me {}.".format(self.name))

@classmethod
def how_many(cls):
    """Prints the current population."""
    print("We have {:d} robots.".format(cls.population))
```

Class And Object Variables

```
droid1 = Robot("R2-D2")
droid1.say_hi()
Robot.how_many()

droid2 = Robot("C-3PO")
droid2.say_hi()
Robot.how_many()

droid1.die()
droid2.die()

Robot.how_many()
```

Class And Object Variables

Output

```
(Initializing R2-D2)
Greetings, my masters call me R2-D2.
We have 1 robots.
(Initializing C-3PO)
Greetings, my masters call me C-3PO.
We have 2 robots.
R2-D2 is being destroyed!
C-3PO is being destroyed!
We have 0 robots.
```

Inheritance

Inheritance

```
class SchoolMember:

    def __init__(self, name, age):
        self.name = name
        self.age = age
        print('(Initialized SchoolMember: {})'.format(self.name))

    def tell(self):
        print('Name:"{}" Age:"{}"'.format(self.name, self.age), end=" ")
```

Inheritance

```
class Teacher(SchoolMember):

    def __init__(self, name, age, salary):
        SchoolMember.__init__(self, name, age)
        self.salary = salary
        print('(Initialized Teacher: {})'.format(self.name))

    def tell(self):
        SchoolMember.tell(self)
        print('Salary: "{:d}"'.format(self.salary))
```

Inheritance

```
class Student(SchoolMember):

    def __init__(self, name, age, marks):
        SchoolMember.__init__(self, name, age)
        self.marks = marks
        print('(Initialized Student: {})'.format(self.name))

    def tell(self):
        SchoolMember.tell(self)
        print('Marks: "{:d}"'.format(self.marks))
```

Inheritance

```
t = Teacher('Ahmad', 40, 1234)
s = Student('Khaled', 25, 75)


members = [t, s]
for member in members:
    # Works for both Teachers and Students
    member.tell()
```

Inheritance

Output:

(Initialized SchoolMember: Ahmad)
(Initialized Teacher: Ahmad)
(Initialized SchoolMember: Khaled)
(Initialized Student: Khaled)
Name:"Ahmad" Age:"40" Salary: "1234"
Name:"Khaled" Age:"25" Marks: "75"

Input and Output

Input from user

```
text = input("Enter text: ")

# Enter text: Ahmad
```

Files

```
poem = '''\
Programming is fun
When the work is done
if you wanna make your work also fun:
    use Python!
'''

# Open for 'w'riting
f = open('poem.txt', 'w')
# Write text to file
f.write(poem)
# Close the file
f.close()
```

Files

```
# If no mode is specified,
# 'r'ead mode is assumed by default
f = open('poem.txt')
while True:
    line = f.readline()
    # Zero length indicates EOF
    if len(line) == 0:
        break
    print(line, end='')
# close the file
f.close()
```

Files

Output

Programming is fun
When the work is done
if you wanna make your work also fun:
*   use Python!*

Exceptions

Errors

- Consider a simple print function call. What if we misspelt print as Print? Note the capitalization. In this case, Python raises a syntax error.

```
Print("Hello World")
# NameError: name 'Print' is not defined

print("Hello World")
# Hello World
```

Exceptions

- Exceptions occur when *exceptional* situations occur in your program. For example, what if you are going to read a file and the file does not exist? Or what if you accidentally deleted it when the program was running? Such situations are handled using **exceptions**.

```
10 * (1/0)
# ZeroDivisionError: division by zero
```

Handling Exceptions

```
try:
    10 * (1/0)
except ZeroDivisionError:
    print("Can't division by zero")

# Can't division by zero
```

Handling Exceptions

```
try:
    x = int(input("Please enter a number: "))
except ValueError:
    print("Oops!  That was no valid number.")

# Please enter a number: df
# Oops!  That was no valid number.
```