SNS COLLEGE OF ENGINEERING
Kurumbapalayam (Po), Coimbatore – 641 107

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

python

# 19IT103 – COMPUTATIONAL THINKING AND PYTHON PROGRAMMING

- A readable, dynamic, pleasant, flexible, fast and powerful language

## Objective

---

**Files and exception:** text files, reading and writing files, format operator; command line arguments, errors and exceptions, handling exceptions, modules, packages; Illustrative programs: word count, copy file, Voter’s age validation, Marks range validation (0-100).

## RECAP

An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.

- Handling Exceptions
- Raising an exceptions
- User-Defined Exceptions

## Modules in Python

## Modules in python

Modules in Python are simply Python files with a .py extension. The name of the module will be the name of the file. A Python module can have a set of functions, classes or variables defined and implemented.

## What are Modules?

- Modules are files containing Python definitions and statements (ex. *name*.py)
- A module’s definitions can be imported into other modules by using “import *name*”
- The module’s name is available as a global variable value
- To access a module’s functions, type “*name.function()*”

## More on Modules

- Modules can contain executable statements along with function definitions
- Each module has its own private symbol table used as the global symbol table by all functions in the module
- Modules can import other modules
- Each module is imported once per interpreter session
  - *reload(name)*
- Can import names from a module into the importing module's symbol table
  - *from mod import m1, m2* (or **)*
  - *m1()*

## Executing Modules

- *python name.py <arguments>*
  - Runs code as if it was imported
  - Setting  *_name_* == "*_main_*" *the file can be used as a script and an importable module*

## The Module Search Path

- The interpreter searches for a file named *name.py*
  - Current directory given by variable *sys.path*
  - List of directories specified by **PYTHONPATH**
  - Default path (in UNIX - *.:/usr/local/lib/python*)
- Script being run should not have the same name as a standard module or an error will occur when the module is imported

# “Compiled” Python Files

- If files *mod.pyc* and *mod.py* are in the same directory, there is a byte-compiled version of the module *mod*
- The modification time of the version of *mod.py* used to create *mod.pyc* is stored in *mod.pyc*
- Normally, the user does not need to do anything to create the *.pyc* file
- A compiled *.py* file is written to the *.pyc*
  - No error for failed attempt, *.pyc* is recognized as invalid
- Contents of the *.pyc* can be shared by different machines

## Some Tips

- -*O* flag generates optimized code and stores it in *.pyo* files
  - Only removes *assert* statements
  - *.pyc* files are ignored and *.py* files are compiled to optimized bytecode
- Passing two –*OO* flags
  - Can result in malfunctioning programs
  - _*doc*_ strings are removed
- Same speed when read from *.pyc*, *.pyo*, or *.py* files, *.pyo* and *.pyc* files are loaded faster
- Startup time of a script can be reduced by moving its code to a module and importing the module
- Can have a *.pyc* or *.pyo* file without having a *.py* file for the same module
- Module *compileall* creates *.pyc* or *.pyo* files for all modules in a directory

# Standard Modules

- Python comes with a library of standard modules described in the Python Library Reference
- Some are built into interpreter
- *>>> import sys*

  *>>> sys.s1*

  *'>>> '*

  *>>> sys.s1 = 'c> '*

  *c> print 'Hello'*

  *Hello*

  *c>*
- *sys.path* determines the interpreters's search path for modules, with the default path taken from **PYTHONPATH**
  - Can be modified with append() (ex. *Sys.path.append('SOMEPATH')*

## The *dir()* Function

- Used to find the names a module defines and returns a sorted list of strings
  - >>> *import mod*

    >>> *dir(mod)*

    *['_name_', 'm1', 'm2']*
- Without arguments, it lists the names currently defined (variables, modules, functions, etc)
- Does not list names of built-in functions and variables
  - Use _*bulltin*_to view all built-in functions and variables

## Why we need them ?

---

To take advantage of modular programming:

Modular programming refers to the process of breaking a large, unwieldy programming task into separate, smaller, more manageable subtasks or modules. Individual modules can then be cobbled together like building blocks to create a larger application

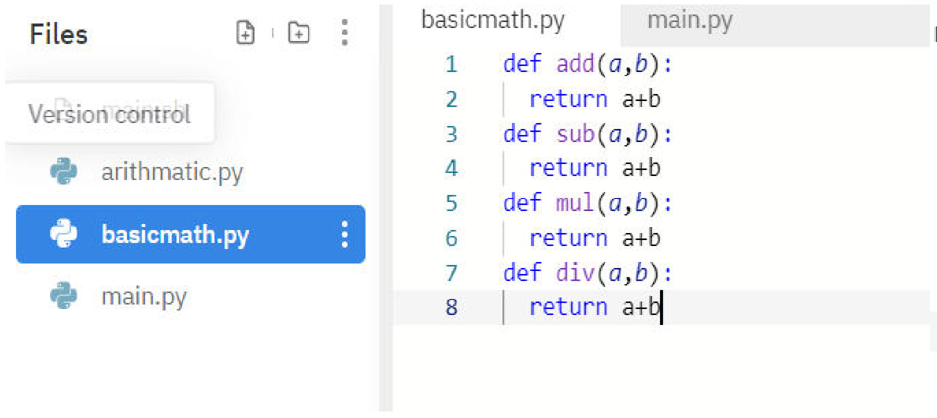
## Advantages

---

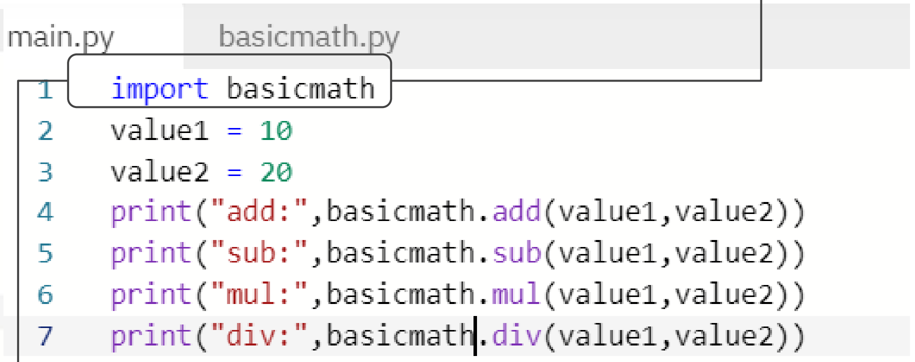
- Simplicity
- Maintainability
- Reusability
- Scoping

# Usage

Main.py using the math module

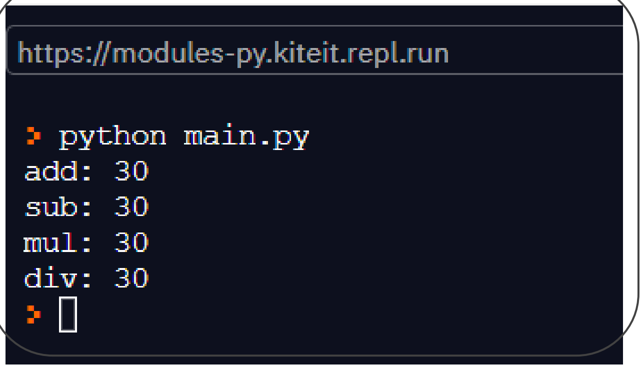


```python
def add(a,b):
  return a+b
def sub(a,b):
  return a+b
def mul(a,b):
  return a+b
def div(a,b):
  return a+b
```



```python
import basicmath
value1 = 10
value2 = 20
print("add:",basicmath.add(value1,value2))
print("sub:",basicmath.sub(value1,value2))
print("mul:",basicmath.mul(value1,value2))
print("div:",basicmath.div(value1,value2))
```

Basicmath.py (module contains useful arithmetic functions)



```
https://modules-py.kiteit.repl.run

> python main.py
add: 30
sub: 30
mul: 30
div: 30
>
```

## SUMMARY

Modules in Python are simply Python files with a .py extension.

- Executing Modules
- Standard Modules
- dir() Function

Thank you