Introduction to Programming Language (ITP101)

Intro to Python

Mulualem Teku

GCIT, Bhutan

Aug 14, 2019

...So Far & Today...

- Computational Thinking: the whats and techniques:
  1. Decomposition
  2. Pattern recognition
  3. Abstraction
  4. Algorithms
- Algorithms: flowcharts / Pseudocodes
- Input/output determination, testing algorithms (trace tables)
- Building blocks of problem solving

Today:

- Programming languages and paradigms
- Programming tools
- Python programming

Brainstorm

1. Data vs information vs knowledge?
2. Programming language(s) used in each generation of computers?
3. Algorithm vs programming? Programming paradigms?
4. Compilers , Interpreters , Assemblers ?

Brainstorm

1. Data vs information vs knowledge?
2. Programming language(s) used in each generation of computers?
3. Algorithm vs programming? Programming paradigms?
4. Compilers / Interpreters / Assemblers ?

Brainstorm

1. Data vs information vs knowledge?
2. Programming language(s) used in each generation of computers?
3. Algorithm vs programming? Programming paradigms?
4. Compilers / Interpreters / Assemblers ?

Brainstorm

1. Data vs information vs knowledge?
2. Programming language(s) used in each generation of computers?
3. Algorithm vs programming? Programming paradigms?
4. Compilers / Interpreters / Assemblers ?

Algorithm-Program Relation

Analogy

- Suppose you have written the recipe for a delicious Ema Datsi in English.
- You want to tell a person in Thailand (who doesn't speak English) how to make it.
- Solution: translate your recipe into thai. You recipe can now be "run" in Thailand.
- { you = programmer, Algorithm = recipe, the thai person = computer }

Algorithm-Program Relation

Analogy

- Suppose you have written the recipe for a delicious Ema Datsi in English.
- You want to tell a person in Thailand (who doesn't speak English) how to make it.
- Solution: translate your recipe into thai. You recipe can now be "run" in Thailand.
- { you = programmer, Algorithm = recipe, the thai person = computer }

Programming Languages

- Algorithms must be translated into sequence of computer instructions by means of **programming languages**.

- Programming language categories:

  1. High-level language: closer to human language (e.g. C, C++, Java, **Python**)
  2. Assembly language: mnemonic/symbolic-based (e.g. ADD, SUB, MOV)
  3. Low-level (machine) language: native language of computer circuitry (0/1)

- Each language has syntactic and semantic rules.

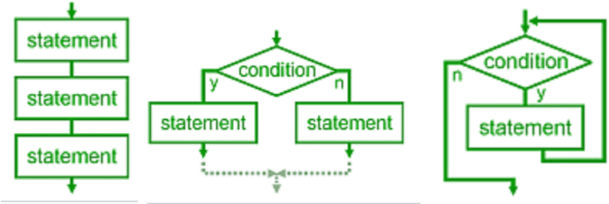
- If not followed, syntax, logical and/or runtime errors.

Programming Paradigm

- Paradigm = pattern/model of something.
- It is a way of classifying programing languages based on *features*.
- Various programming paradigms exist. Some are:
  1. Structured paradigm
  2. Procedural paradigm
  3. Object-oriented paradigm
  4. Logical paradigm
  5. Functional paradigm

Structured Paradigm

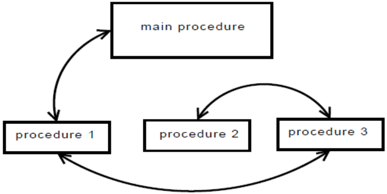
- Programs composed of control structures / flow statements.
- Makes extensive use of structured control flow statements (if...else, loops), blocks, and subroutines.
- 3 basic patterns are: sequences, selection and iteration.



E.g. Python, C, ALGOL

Procedural Paradigm

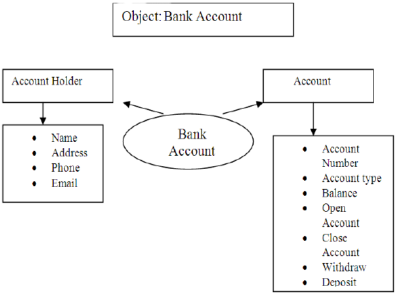
- Program is modeled as a sequence of procedure/function calls.
- Variables represent memory locations.
- Flow of execution can be traced from a statement.
- The focus is to break down a task into a set of variables, data structures and subroutines.



E.g. C, Python, FORTRAN

Object-oriented Paradigm

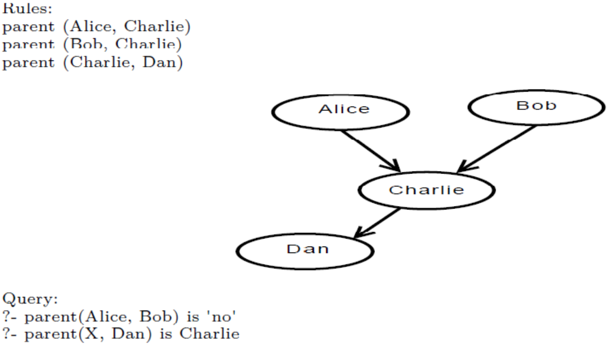
- Program modeled as a set of objects with **state/attributes** and **behavior/methods**.
- Attributes define the object. Methods define actions that can be performed on the object.



E.g. Java, C++, Python, Smalltalk

Logical Paradigm

- Computer viewed as a logical inference device.
- Program modeled as a set of logical rules/statements.



Rules:
parent (Alice, Charlie)
parent (Bob, Charlie)
parent (Charlie, Dan)

Query:
?- parent(Alice, Bob) is 'no'
?- parent(X, Dan) is Charlie

E.g. Prolog

Python

- Not the reptile $\ddot\smile$
- Writen by Guido van Rossum (GvR) - the BDFL.
- Named after a British TV show: Monty Python's Flying Circus.
- A *modern*, *interpreted*, *object-oriented* and *versatile* language.
- Multi-paradigm language:
  mainly procedural, object-oriented *&* functional.
- Versions: Python 2.x, Python 3.x
- Implementation flavors: `CPython`, `Jython`, `IronPython`, `PyPy` ...

Python: Salient Features

- Free
- Low learning curve
- Portable, Powerful
- Extensible/ Support Libraries (modules)

- Development Speed
- Component Integration
- Object-Oriented
- Dynamic Memory Mgmt

Python: Userbase

Google app engine

PIXAR
Animation Studios

National Security Agency
United States of America

Python: Application Domains

- Internet Programming

  Standard Internet modules (e.g. network programming) + web-development frameworks (e.g. `Web2py`, `Django`, `TurboGears`)

- Database Programming

  Interfaces to relational DBMSs such as MySQL, Oracle, Sybase...

- GUIs (e.g. `Tkinter`, `wxPython`), gaming (e.g. `pygame`)

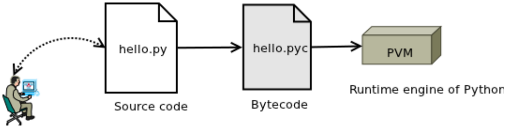
- Scientific Computing (e.g. `Numpy`, `Scipy`, `Pandas`, `Matplotlib`)

- Systems Programming

  Standard libraries for OS interfaces (files, processes, sockets, etc)

The Execution Model

Python is a dynamic, *interpreted* language.



- **Bytecode** is a lower-level, *platform-independent* representation.
- If only source changes is the code ‘recompiled’.
- Otherwise, `.pyc` file loaded and run ⇒ an optimization mechanism!
- The **Python Virtual Machine (PVM)** executes the bytecode one-by-one, using the CPU’s architecture-specific instructions.
- Everything happens at runtime.

Python: Getting Started

- Running Modes

  1. Interactive mode

     ```
     $ Python
      >>> print ("Hello World!")        # this is comment
     ```

  2. Script mode

- Error handling

- Editors and/or IDEs → `Notepad, Vi/Vim, IDLE, Spyder ...`

- Interactive Python interpretter → `IPython`

- Everything in Python is an object.

Core Python Objects

- Numbers
- Boolean
- Functions
- Strings
- Lists
- Tuples
- Dictionaries
- Files
- Modules
- Classes

- Mutable vs Immutable objects

Data Types: 1) Numbers

Basic Numeric Types

- Integer (normal & Long)

  e.g. 25, 65535, 9999999999999999999

- Float

  e.g. 3.14, 2.17e-30, 6.02E+23

- Complex (RealPart + ImaginaryPart)

  e.g. 2+5j, complex(0,9)

Data Types: 1) Numbers

Basic Numeric Types

- `Integer` (normal & Long)

  `e.g.   25, 65535, 9999999999999999999`

- `Float`

  `e.g.  3.14, 2.17e-30, 6.02E+23`

- `Complex` (RealPart + ImaginaryPart)

  `e.g.  2+5j, complex(0,9)`

Conversion functions → `int()`, `float()`, `long()`, `bin()`, `hex()`

Common math functions are used by *importing* the `math` module/library.

e.g. `sqrt()`, `pow()`, `min()`, `sin()`, `floor()`, etc

Operators

An operator is a symbol that represents an operation (e.g. +, *, !=).

- Arithmetic operators

  `+` `-` `*` `/` `//` `%` `**`

- Relational operators

  `<` `>` `==` `<=` `>=` `!=` `<>`

- Logical operators

  `and` `or` `not`

- Bitwise operators

  `<<` `>>` `&` `|` `^` `~`

Variables

- Variable is simply a name (identifier) associated with a value.

- No need to declare variables but need to initialize (using assignment operator i.e. '='. Notice the difference between '==' and '=').

- Python is a dynamically-typed language.

```
>>>var = "Hello World"

>>>var = 1234

>>>print var
```

Identifiers

- A sequence of one or more characters/letters used to name a variable.

Identifier naming rules

- First character be a letter; the rest can be any number of alphanumeric or (_).
- Case-sensitive.
- Not be a reserved (key) word such as 'if', 'for', 'def', 'import', etc.
- No special characters allowed.

Examples

(valid): `totalPrice, firstName, total_price, totalPrice2019`

(invalid): `"totalPrice", first name, _totalPrice, 2019totalPrice`

Identifiers

- A sequence of one or more characters/letters used to name a variable.

Identifier naming rules

- First character be a letter; the rest can be any number of alphanumeric or (_).
- Case-sensitive.
- Not be a reserved (key) word such as 'if', 'for', 'def', 'import', etc.
- No special characters allowed.

Examples

(valid): `totalPrice, firstName, total_price, totalPrice2019`

(invalid): `"totalPrice", first name, _totalPrice, 2019totalPrice`

Keywords

- A keyword is an identifier that has predefined meaning in a programming language.
- Not to be used as variable identifiers.

Python keywords

`and`	`as`	`assert`	`break`	`class`	`continue`	`def`	
`elif`	`else`	`except`	`finally`	`for`	`from`	`global`	`if`
`import`	`in`	`is`	`lambda`	`nonlocal`	`not`	`or`	`pass`
`raise`	`return`	`try`	`while`	`with`	`yield`		

Expressions

- An expression is a combination of symbols that evaluates/reduces to single value.
- Expression = operator(s) + operand(s).
- Be mindful of operator precedence when evaluating expressions.

Examples:

```
7.0 // 3

5 * 2 ** 3

5 + 2 * 14

6 / (10 +  3) * (4 - 8)
```

Statements

- Statements are instructions that a Python interpreter executes.

- Express some action to be carried out.

- Program = collection of one ore more statements (could include expressions).

  Some examples:

  - Assignment statements (=)
  - Loop statements (‘for’ and ‘while’ loops)
  - Conditional statements (`if...else`)

Modules & Imports

- Python programs are composed of modules.
- Modules contain statements.

  e.g. `hello.py` (source code) → a.k.a. module “hello”

Module

A file containing Python definitions & statements.

- Each module be *imported* to be used.

  `import <module name>`

- Some standard modules
  - `math` - math functions
  - `sys` - access to `exit()`, `stdout`, `stdin`, `argv` ...
  - `os` - file system, operating system interface, etc

Modules & Imports

- Python programs are composed of modules.
- Modules contain statements.

  e.g. `hello.py` (source code) → a.k.a. module “hello”

Module

A file containing Python definitions & statements.

- Each module be *imported* to be used.

```
    import <module name>
```

- Some standard modules
  - `math` - math functions
  - `sys` - access to `exit()`, `stdout`, `stdin`, `argv` ...
  - `os` - file system, operating system interface, etc

Console Input/Output

- **Console Output**

```
>>>print "Hi there"                                       # Python 2.x

>>>print ("Hi there")                                     # Python 3.x

>>>import sys
>>>sys.stdout.write("Hi there")                           # try it
```

- **Console Input**

```
>>>msg = input("Enter a message:")

>>>print msg
```

For numeric inputs, the response be converted to the appropriate type (e.g. int, float) using the built-in type conversion functions (e.g. `int()`, `float()`)

Console Input/Output

- **Console Output**

```
>>>print "Hi there"                                    # Python 2.x

>>>print ("Hi there")                                  # Python 3.x

>>>import sys
>>>sys.stdout.write("Hi there")                        # try it
```

- **Console Input**

```
>>>msg = input("Enter a message:")

>>>print msg
```

For numeric inputs, the response be converted to the appropriate type (e.g. int, float) using the built-in type conversion functions (e.g `int()`, `float()`)

Console Input/Output

- **Console Output**

```
>>>print "Hi there"                                   # Python 2.x

>>>print ("Hi there")                                 # Python 3.x

>>>import sys
>>>sys.stdout.write("Hi there")                       # try it
```

- **Console Input**

```
>>>msg = input("Enter a message:")

>>>print msg
```

For numeric inputs, the response be converted to the appropriate type (e.g. int, float) using the built-in type conversion functions (e.g. `int()`, `float()`)

Examples

1.
2.
3.
4.

Getting Help

- The built-in help() function

Example

```
>>>help(‘modules’)      # list of available modules

>>>import math

>>>help(math)           # docs for ‘math’ module

>>>help(math.sin)       # doc for ‘sin()’ of ‘math’ module

>>>dir(math)           # defined members in ‘math’ module
```

- The official Python documentation (www.python.org) is an authoritative source of reference.

Next...

Control Structures / Flow Control

Brainstorm

1. What are statements in programming? Name some statements.

2. Recall our discussion on algorithms/flowcharts. Some flowchart symbols?

3. Design a flowchart to:

    1. Determine if a given number is even or odd.
    2. Display “Kuzu zangpo” 20 times.
    3. Display even numbers between 1 and 20.

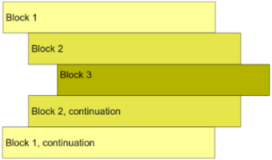
Control Structures

- Recall programming paradigms. Structured paradigm?
- Control Structures are statements that dictate the flow of program.
- 3 basic control structures:
  1. Sequential (default mode).
  2. Selection/branching/decisions.
  3. Repetition/iteration/loops

Block Statements (Python)

- A block in programming is a group of statements grouped together and treated as if a single statement.

- Different programming languages use various ways to indicate code blocks. Python uses indentation.



Block 1

Block 2

Block 3

Block 2, continuation

Block 1, continuation

- Before we look at control structures, remember:

  `True` ≡ any non-zero or non-empty objects.

  `False` ≡ zero, empty object, or `None`.

Flow Control: The `if` Statement

- Two-way branching

```
if <test>:                          # notice the colon
    <statements 1>                  # the indentation too
else:                               # notice the colon
    <statements 2>                  # else is optional
```

- The Ternary Expression

  e.g.   `max = x if (x > y) else y`

- Multi-way branching   `if...elif`
  - No `switch` statement in Python .

Flow Control: The `if` Statement

- Two-way branching

```
if <test>:                          # notice the colon
    <statements 1>                  # the indentation too
else:                               # notice the colon
    <statements 2>                  # else is optional
```

- The Ternary Expression

  e.g. `max = x if (x > y) else y`

- Multi-way branching `if...elif`
  - No `switch` statement in Python .

Flow Control: The `if` Statement

- Two-way branching

```
if <test>:                         # notice the colon
    <statements 1>                 # the indentation too
else:                              # notice the colon
    <statements 2>                 # else is optional
```

- The Ternary Expression

  `e.g.    max = x if (x > y) else y`

- Multi-way branching `if...elif`
  - No `switch` statement in Python .

Example

```
n = 101
if  n % 2 == 1:
    print ("Odd")    # notice the indentation
else:
    print ("Even")
```

Example

```
x = float(input("Enter magnitude of the earth quake:") )

if x < 4.0:
    print ("Minor")
elif 4.0 < x < 6.0:
    print ("Moderate")
else:
    print ("Major")
```

Example

```
n = 101
if  n % 2 == 1:
    print ("Odd")    # notice the indentation
else:
    print ("Even")
```

Example

```
x = float(input("Enter magnitude of the earth quake:") )

if x < 4.0:
    print ("Minor")
elif 4.0 < x < 6.0:
    print ("Moderate")
else:
    print ("Major")
```

Examples

1.
2.
3.

Flow Control: for Loops

```
for <iterating_var> in <sequence object>:
      <statements>
```

- Must be indented! Notice the colons.
- `<iterating_var>` is an iterating variable of our choice.
- `<sequence object>` can be strings, lists, tuples & dictionaries. More on these objects later on.

```
>>>for i in [2, 4, 6, 8, 10]:
      print (i)
```

```
>>>for j in "Hello world":
      print (j)
```

Flow Control: for Loops

```
for <iterating_var> in <sequence object>:
      <statements>
```

- Must be indented! Notice the colons.
- `<iterating_var>` is an iterating variable of our choice.
- `<sequence object>` can be strings, lists, tuples & dictionaries. More on these objects later on.

```
>>>for i in [2, 4, 6, 8, 10]:
       print (i)
```

```
>>>for j in "Hello world":
       print (j)
```

The Range() Function

- Generates lists containing arithmetic progressions.
- `range(start, end, step)`

```
>>>range(10)                       # start = 0, last = end-1

>>>range(2,20,2)

>>>range(50,0,-1)

>>>range(0,9,5)

>>>for i in range(1, 10):
        print (i**2)
```

Examples

1. Display numbers 1 to 100.

2. $$\sum_{n=1}^{100} n$$

3. $$\prod_{i=1}^{n} \sqrt{i}$$ `# user-supplied n`

Examples

1. Display numbers 1 to 100.

2. $$\sum_{n=1}^{100} n$$

3. $$\prod_{i=1}^{n} \sqrt{i}$$ `# user-supplied n`

Examples

1. Display numbers 1 to 100.

2. $\sum_{n=1}^{100} n$

3. $\prod_{i=1}^{n} \sqrt{i}$ `# user-supplied n`

Iterations: while Loops

```
while <test>:
      <some statements>
```

```
i = 1
while i < 5:
     print ("Kuzu zangpo")
     i = i + 1
```

```
n = 1
while n < 5:
     print (n)
     n = n + 1
print ("Bye")
```

Example

Display numbers 100 to 1.

Example

What does the following code snippet achieve?

```
i = 1
result = 0
while i <= 10:
      result = result + i
      i = i + 1
print result
```

- The break and continue statements

Example

What does the following code snippet achieve?

```
i = 1
result = 0
while i <= 10:
      result = result + i
      i = i + 1
print result
```

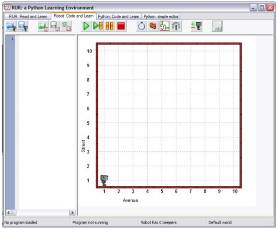
- The `break` and `continue` statements

Examples

1.
2.
3.

Fun with Python

- RUR-PLE - is an environment designed to help you learn computer programming using Python. Reeborg, a robot, will be at your service. You will be his master. Cool, huh? Make it do a task.



No program loaded

Program not running

Robot has 0 beepers

Default world

- The Python Challenge game, set of programming riddles solved using Python.