PYTHON – Structural Programming

***The process of breaking large, complex task into small subtasks,   
 simple enough to be performed with sequences of instruction*—**

1. Identify - (i) Constant/ Given states, (ii) Desired result, (iii) Math/ Operation  
    (iv) Criteria/ Conditions of execution, and (v) Repetition.
2. Script the pseudo-code as comment.
3. Conduct debugging: unit testing on errors & input exceptions.

## Data Formats & Objects

In Python, object is an instance of a particular type with value & identity.

* *dtype*, (ie. object’s class) decribes the object internal representation as well as the methods &   
   operations it supports. [# shared sets of functions, algorithms, instances]
* *value(s)*, (ie. attr/ element) the int. / ext. properties of an object regarding its type.
* *identity*, (ie. variable), acts as a pointer to an object’s location in memory.

**∴** The value of a mutable obj can be partially changed, but its identity & type can only be re-defined.

Hence, it is important to distinguish the object to efficiently organize the data for its use cases.

**Python’s keywords:** [# *A good practice to use comment for enlightening short variable names* ]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| and | as | assert | break | class | continue |
| def | del | elif | else | except | import |
| finally | for | from | global | if | or |
| in | is | lambda | nonlocal | not | with |
| pass | raise | return | try | while | await |
| yield | True | False | None | async |  |

The naming conventions for identifiers following the Python Style Guide as:

Class – CamelCase*capitalize each word* , Function – make\_payment*a verb*

Attr/ Parameter/ Variable – price*lowercase* , Constant – MAX\_SIZE *all capital letterl*

**Variable scope,** ie. environment/ namespace :

{ Local } — namespace variables declared within a function definition,   
 which create when a function executes and limited only to the space & lifetime.

{ Global } — sessional namespace variables declared outside function bodies, or universally  
 by ‘global’ keyword within a local namespace.

**Note:** Both variables declaration can have the same name w/o conflict, unique to its scope of reference.

*However,*  when a function is called, Python Interpreter looks into: [# Local > Global > Built-in]

So, (1) the variables that are referenced inside a function are global implicitly,

(2) as function definition is imported & read before it is called, any variable assignment   
 within a local namespace will make the variable’s object locally-bounded.

⋙ var = 10 ; var = 10

⋙ def my\_function(): def my\_function():

print(var) print(var)

⋙ my\_function() var = var + 1

::10 my\_function()

::UnboundLocalError: local variable ‘var’ referenced before assignment

∴ both global & local variables cannot point to the same object.

‘global’ should only be used to declare variables of a sytem’s states from within the local scope, and otherwise may w/ global identity at associated project updates.  
[ —construct OOP (classes) instead, if object is required to be called regularly ]

## Conditions

Here we see a different algebra – the *Boolean* algebra – which provides rules for working   
with Boolean values.

**(A)** ‘and’ operator: **(B)** ‘or’ operator: **(C)** ‘not’ operator:

x and False ≃ False x or False ≃ x not (not x) ≃ x

False and x ≃ False False or x ≃ x

y and x ≃ x and y y or x ≃ x or y

x and True ≃ x x or True ≃ True

True and x ≃ x True or x ≃ True

x and x ≃ x x or x ≃ x

(Simplification Laws)

**READING :** 1. break large, complex task into

**A PROGRAM** smaller subtasks, identify the tokens,

and interpret the structure.

(eg. *tokens, semantics » syntax » structure*)

2. Details matters.

(eg. *punctuations, indentations, continuity*)

not (x and y) ≃ (not x) or (not y)

not (x or y) ≃ (not x) and (not y)

ie. if not ((x >= 0.9) and (y >= 100)):

*helpful for dealing w/ complicated ‘not’*,

if (x < 0.9) or (y < 100):

# Functions.

* It presents as a black box tool which provides regular functionality to users   
  based on *argument*s & *expected results*, independent of the accuracy from assumptions.

**Variables**: mandatory arg(s)   
 of generalization

**Options:** altering function/   
 parameters setting

**Packed/Extensible args**   
not initially expected in fn.

def NAME(VARS: str, OPTIONS=default, \*ARGS/\*\*KWARGS):

‘‘‘DOCSTRINGS’’’

STATEMENTS

return VALUE1, VALUE2

\**type hints/annotations only trigger warning pops up, not forcing datatypes as inputs.*

(Utility of function creation):

1. To facilitate the use of iteration.
2. Dividing long program to independent components for isolated debugging environment.

[ *Docstring*— provide descriptions of functionality, arguments & result as ref in - help(fn)/fname?]

**Note:**

1. Ensure every possible path through a fruitful function hits a return -statement.
2. Conforming to the conduct of local namespace, subfunctions are only accessible to class / module   
    within which it resides. (*ie*. via import statement)
3. Passing a mutable object as function argument actually creates an alias  
   (*ie*. any changes w/o copy should be explicitly documented in docstring as modified in-place)

***Example***. **# Function block** **# Recursive function**

def clean\_strings(strings): def recurTest(low, high):

result = [] if low <= high:

for value in strings: print(low)

value = value.strip() recurTest(low+1, high)

value = re.sub('[!#?]', '', value) **# Iterative function**

value = value.title() def iterTest(low, high):

result.append(value) while low <= high:

return result print(low)

low += 1

[# In general, iterative is more efficient, but recursive are useful for manipulating recursive data structures   
—linked lists & trees (Refer. Recursive data structure) ]

Python always wants to return something,

{**Fruitful function**}— return value(s)

{**Void function**}— return *None* object

{**Boolean function**}— return True/False

@ Reminder:

(1) Do not use the following statements   
 inside function (*unless debugging*)  
 - Print ; User Input ; File I/O ; Pause ; Plot/Graphic

(2) Boolean function is convenient for hiding complicated test and should allocate with [ yes/no ] question names.