Python

Introduction to User-defined functions, type casting, operators, errors

User defined functions

Definition

```
def fname(formal parameter list):
    body of function
    composed of indented
    statements
    return r_expr
```

Usage (calling or invoking functions)

```
fname(actual parameter list)
```

can be used in expressions

Function's definition

fname: name of function

parameter list: comma separated list of variable names

return: reserve word

(ends execution of functions body)

rexpr: expression to return to caller statement

return line may be omitted completely or rexpr is not mandatory (None return is both cases)

Functions

```
// to compute and return average
// of three integer parameters
def average(a, b, c):
      sum = a+b+c;
      avg = sum/3;
      return avg;
```

Using a function

```
print(average(23, 7, 6))
```

Situation (type mismatching)

The output of the following simple code segment given after it, is not what we are expecting.

```
i = 13
b = True
f = 2.5
r = (i-b)/ f
print("(13 - True) / 2.5 is ", r)
```

```
(13 - True) / 2.5 is  4.8
```

Type Conversion and Type Casting?

Type castings

- Implicit type casting
  - Shorter sized data types are promoted to longer sized data types in expressions of operands with mismatched (but appropriate) data types.
  - No lose of data
- Explicit type casting
  - Programmer specified promotion or demotion of values for operands before being used in expression.
  - Type Casting functions: int(), str(), bool(), …
  - Data may loose, but on programmers wish

Operators

- Categories
  - Arithmetic
  - Relational
  - Identity and membership
  - Logical or Boolean
  - Bitwise
  - Assignment
  - Etc, etc

PEDMAS

- Precedence
  - Which operation will perform earlies than other in same expression

- Associativity
  - Same precedence operation will operate left to right or right to left

Common operators

Operators	Assoc	Description
		Non Category
()	--	Parentheses (grouping), function call
[]	--	indexing, slicing
.	--	attribute reference
		Arithmetic
+ -	RL	unary versions
**	RL	power or exponent
* / // %	LR	multiplication and division
+ -	LR	addition and subtraction
		Conditional
< <= > >=	LR	inequality relational
== !=	LR	equality relational
is,in,not is,not in	LR	identity and membership
		Logical
not	RL	logical NOT
and	LR	logical AND
or	LR	logical OR
		Assignment
=	RL	assignment
*=	RL	multiplication and assignment
/=	RL	division and assignment
%=	RL	modulus (remainder) and assignment
+=	RL	addition and assignment
-=	RL	subtraction and assignment

Overloaded operators

Operator are defined for types other than Numbers and Booleans

e.g.

- `str + str`
- `str * number`
- `str [not] in str`
- `list * number`
- `list + list`

Errors

- Syntax errors
  - If exist, compiler/interpreter is unable to get the statement recognized and interpret it, and you have no choice but to remove them.
- Warnings (usually in compilers)
  - May tell you about your possible logical errors, program can run with them. MUST understand them and remove them, if these WARNINGS are harmful
- Logical Errors
  - If exist, behavior of the program is unexpected, wrong results, missing values, infinite execution
- Runtime errors / Exceptions
  - Occurs when data is not appropriate for an operation, e.g. divide by ZERO, crossing arrays limits
- Exceptions
  - Raised by the code through special instructions

Arrays

var = [initval] * size

or

var = [expr_list]

var is array name, size is size of the array, and initval is the ***initial value*** filled in each cell of array

Each cell is accessible with an integer index starting from 0 to size-1, e.g., to access the 3rd cell of array named var, we use var[2]

expr_list is list of comma separated expressions

Example

```
# to compute and return average of
# whole integer array with N values

def average(a, N):
      sum = 0
      j = 0
      while j < N:
            sum = sum+a[j]
            j = j+1

      avg = sum/N
      return avg
```

```
def main():
      v = [5,3,4,7,4]
      av = average(v,5)
      print(av)
      return
```