C

Chasing the Bugs

"Wading through the choppy waters..."

KanNotes

How can we chase away the bugs in a Python program? No sure-shot way for that. So I thought if I make a list of more common programming mistakes, it might be of help. I have presented them below. They are not arranged in any particular order, but I think, they would be a good help!

Bug 1

Mixing tabs with spaces in indentation.

Consider the code snippet given below:

```
if a < b :
    a = 10
    b = 20
```

Here the first statement in if block has been indented using tab, whereas the second has been indented using spaces. So on the screen the snippet looks alright, but Python interpreter will flag an error. Such errors are difficult to spot, so always use 4 spaces for indentation.

Bug 2

Missing : after if, loop, function, class.

Since other languages do not need a : those who migrate to Python from other languages tend to forget to use :.

Bug 3

Using ++ or --.

Don't increment/decrement using ++ or --. There are only two ways to increment/decrement a variable:

```
i = i + 1
i += 1
```

Bug 4

No static types for variables.

Unlike other languages, we do not have to define the type of the variable. Type of the variable is determined dynamically at the time of execution based on the usage of the variable. So in the following code snippet **a** is integer to begin with, but when the context changes its type changes to str.

```
a = 25
print(type(a))        # prints <class 'int'>
a = 'Hi'
print(type(a))        # prints <class 'str'>
```

Bug 5

Deleting an item from a list while iterating it.

```
lst = [n for n in range(10)]
for i in range(len(lst)) :
if i % 2 == 0 :
    del lst[i]
```

Correct way to do this is to use list comprehension as shown below:

```
lst = [n for n in range(10)]
lst = [n for n in lst if n % 2 != 0]
print(lst)
```

Bug 6

Improper interpretation of **range( )** function.

Remember the following for loop will generate numbers from 0 to 9 and not from 1 to 10.

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

Bug 7

Using = in place of ==.

When performing a comparison between two objects or value, you just use the equality operator (==), not the assignment operator (=). The assignment operator places an object or value within a variable and doesn't compare anything.

Bug 8

Difference in built-in and other types while referring to objects.

```
i = 10
j = 10
a = 'Hi'
b = 'Hi'
x = [10]
y = [10]
print(id(i), id(j), id(a), id(b), id(x), id(y))
```

id( ) returns the address stored in its argument. Since **i** and **j** are referring to same int, they contain same address. Since **a** and **b** are referring to same string, they contain same address. However, addresses stored in **x** and **y** are different as two objects each containing [10] are created.

Bug 9

Using improper case in logical values.

All keywords and operator (like **and**, **or**, **not**, **in**, **is**) are in small-case, but logical values are **True** and **False** (not true and false).

Bug 10

Improper order of function calls.

While creating complex Python statements we may place function calls in wrong order producing unexpected results. For example, in the following code snippet if we change the order of the function calls, we get different results.

```
s = "  Hi  "
print(s.strip().center(21, "!"))          # prints !!!!!!!!!!Hi!!!!!!!!!
print(s.center(21, "!").strip( ))         # prints !!!!!!!!  Hi  !!!!!!!
```

Remember that Python always executes functions from left to right.

Bug 11

Improperly initializing a mutable default value for a function argument.

Consider the following code snippet:

```
def fun(lst = [ ]) :
    lst.append('Hi')
    print(lst)

fun( )        # prints ['Hi']
fun( )        # prints ['Hi', 'Hi']
```

It may appear that during each call to fun 'Hi' would be printed. However, this doesn't happen since the default value for a function argument is only evaluated once, at the time that the function is defined. Correct way to write this code would be:

```
def fun(lst = None) :
  if lst is None :
      lst = [ ]
  lst.append('Hi')
  print(lst)

fun( )
fun( )
```

Bug 12

Common exceptions.

Following is a list of common exceptions that occur at runtime and the reasons that cause them:

AssertionError - It is raised when the assert statement fails.

```
age = int(input('Enter your age: '))
assert age >= 0, 'Negative age'
```

AttributeError - It is raised when we try to use an attribute that doesn't exist.

```
s = 'Hi'
s.convert( )# str doesn't have convert( ) method
```

EOFError - It is raised when the input() function hits the end-of-file condition.

ImportError - It is raised when the imported module is not found.

IndexError - It is raised when the index of a sequence is out of range.

```
lst = [10, 20, 30]
print(lst[3])
```

KeyError - It is raised when a key is not found in a dictionary.

KeyboardInterrupt - It is raised when the user hits Ctrl+c.

MemoryError - It is raised when an operation runs out of memory.

NameError - It is raised when a variable is not found in the local or global scope.

RuntimeError - It is raised when an error does not fall under any other category.

StopIteration - It is raised by the **next( )** function to indicate that there is no further item to be returned by the iterator.

TypeError - It is raised when a function or operation is applied to an object of an incorrect type.