Syntax and Control Flow

Of Python3 and C++

By Ariel Parra

Simplest Python Template for input output program

```python
def main():

    n = int(input()) # delcares a variable 'n' and asign an integer from standart input
    print(n) # writes the variable 'n' followed by a newline '\n' to standart output

main() # calls the main function to run
```

Simple Python Template for competitive programming

```python
import sys # imports fast input/output library
read = sys.stdin.readline # fast input function
write = sys.stdout.write # fast output function

def main():

    n = int(read()) # delcares a variable 'n' and asign an integer from standart input
    write(f"{n}\n") # writes the variable 'n' followed by a newline '\n' to standart output

main()
```

> Try the code on https://www.onlinegdb.com/

Simple C++ template for competitive programming

```cpp
#include <bits/stdc++.h> // includes all librarys in c++
using namespace std; // std::cout<<"hi"; -> cout<<"hi";

int main(){
ios::sync_with_stdio(0);cin.tie(0); // enables fast input/output

    int n;       // variable to store the input number
    cin >> n;    // reads an integer 'n' from standard input
    cout << n;   // writes the integer to standard output

return 0;
}
```

Conditionals

Truth Tables for Boolean (Logical) Operators

Py `not`, C++ `!`

p	¬p
F	T
T	F

Py `and`, C++ `&&`

p	q	p ∧ q
F	F	F
F	T	F
T	F	F
T	T	T

Py `or`, C++ `||`

p	q	p ∨ q
F	F	F
F	T	T
T	F	T
T	T	T

simple if's

Python

```python
programming, studying = True, True
if programming:
    write("True\n")  # output with endline '\n'
else:
    write("False\n")
```

C++

```cpp
bool programming, studying = true;
if(programming) //asumes if(programming == true)
    cout << "true\n"; // output with endline '\n'
else //asume if(programming == false) or if(!programming)
    cout << "false" << endl; // output with endl
```

Ternary operator

First, the boolean condition is written, followed by the `?` operator, which indicates the value that will be returned if the condition is true. After the `:` operator, the value that will be returned if the condition is false is placed.

py

```
write("Learning " if programming else "procrastinating") # Py
```

c++

```
cout << (programming ? "Learning " : "procrastinating"); //C++
```

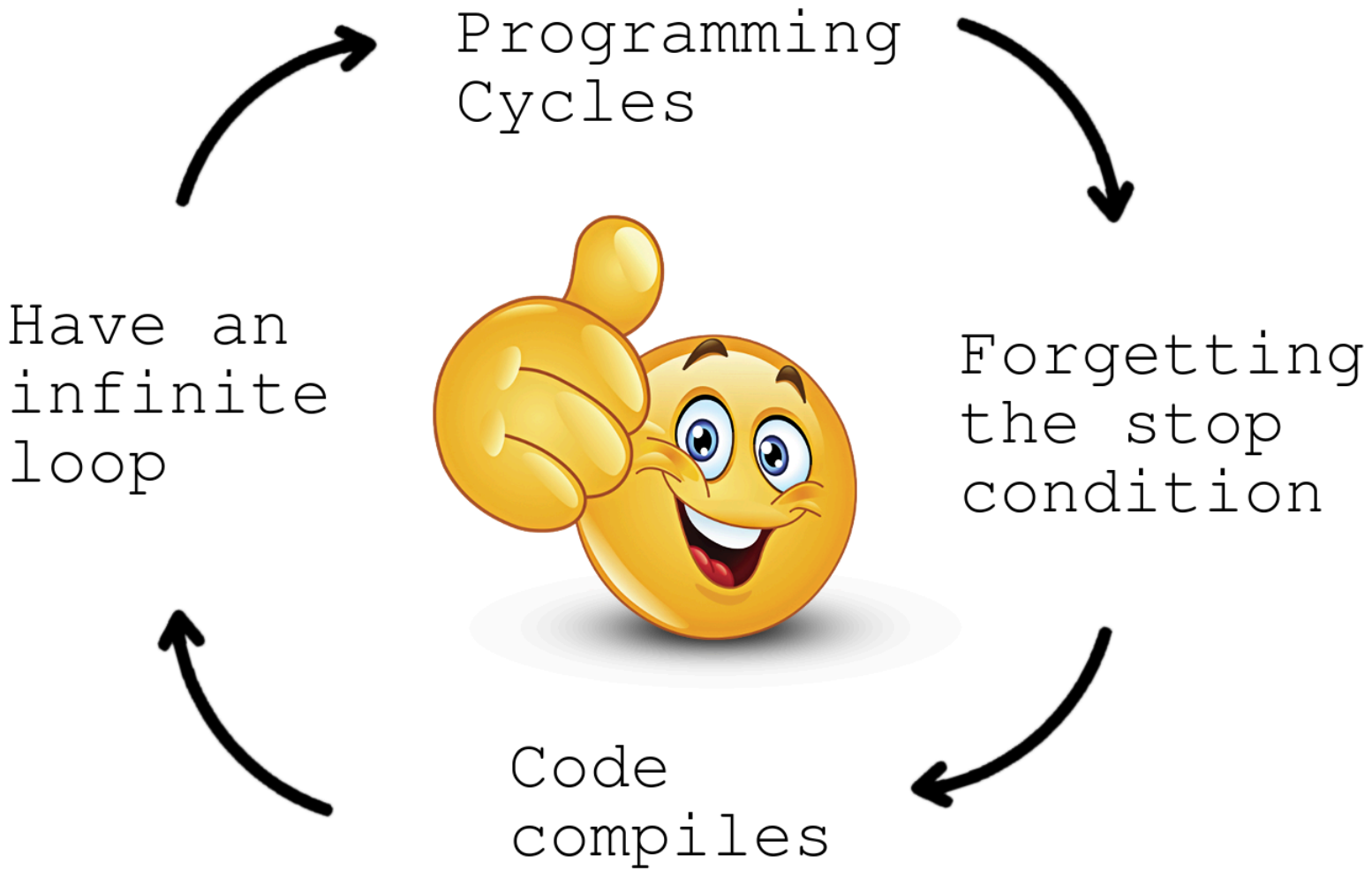
```
cout << (programming && studying ? "I will succeed in competitions"
 : (programming || studying ? "maybe I will succeed" : "well I won't succeed"));
```

Switch-case in c++

It allows evaluating an expression and executing different code blocks efficiently. In this case, `a` is the evaluated expression and can only be numeric (`int` / `long long`) or a character (`char`). Note that Python doesn't have a switch-case statement, so you would need to use if-else statements instead.

```
switch(a) {
    case 'A': case '1':
        cout << "just chars 'A' and '1'";
        break;
    case 'a' ... 'z':
        cout << "lowercase lettters";
        break;
    case 0 ... 10:
        cout << "numbers 0 to 10";
        break;
    default:
        cout << "everything else";
}
```

Cycles


Programming Cycles
Forgetting the stop condition
Code compiles
Have an infinite loop

for

- traditional for

```python
for i in range(n):
    write(f"{i} ")
```

```cpp
for (int i=0; i < n; ++i) {
    cout << i << " ";
}
```

> **Output if n=10**: 0 1 2 3 4 5 6 7 8 9

- single-line for

```cpp
for (int i=0; i < n; ++i) cout << i << " ";
```

```python
[write(f"{i} ") for i in range(n)]
```

- range for

```cpp
vector<int> numbers = {1, 2, 3, 4, 5};
for (int number : numbers) {
    cout << number << " ";
}
```

```python
numbers = [1, 2, 3, 4, 5]
for number in numbers:
    write(f"{number} ")
```

while

- while

```python
i = 0
while True:  # condition
    write(f"{i}")
    i += 1
    if i > n:
        break  # instead of true
```

```cpp
int i = 0;
while (true) { // condition
    cout << i;
    if (i++ > n) break; //insted of true contition
}
```

- do-while (Python doesn't have do-while)

```python
i = 0
write("\n")
while True:
    i += 1
    write(f"{i}")
    if not (i < n):
        break
```

```cpp
int i = 0;
cout<<endl;
do {
    cout << ++i;
} while (i < n);
```

Math sequences

arithmetic sequence

for example: 5,8,11,14,...

py

```
a, d, n = 5, 3, 10
for i in range(n):
    write(f"{a + i * d} ")  # prints all until n term
```

c++

```
int a = 5, d = 3, n = 10;
for (int i = 0; i < n; ++i) {
    cout << a + i * d << " ";//prints all until n term
}
```

geometric sequence

for example: 5,15,45,135,...

```python
a, r, n = 5, 3, 10
for i in range(n):
    write(f"{a * (r ** i)} ")  # prints all until n term
```

```cpp
int a = 5, r = 3, n = 10;
for (int i = 0; i < n; ++i) {
   cout << a * pow(r, i) << " "; // prints all until n term
}
```

Arithmetic sequences

- **Formula for the nth term:**

$$a_n = a_1 + (n-1) \cdot d$$

```
int an = a1 + (n - 1) * d;
```

- **Formula for the sum of the first n terms:**

$$S_n = \frac{n}{2} \cdot (2a_1 + (n-1) \cdot d)$$

```
int sn = ( n * (2 * a1 + (n - 1) * d) ) /2;
```

- or equivalently:

$$S_n = \frac{n}{2} \cdot (a_1 + a_n)$$

Geometric sequences

- **Formula for the nth term:**

$$a_n = a_1 \cdot r^{(n-1)}$$

```
    int an = a1 * pow(r, n - 1);
```

- **Formula for the sum of the first n terms:**

For `r != 1`:

$$S_n = a_1 \cdot \frac{1 - r^n}{1 - r}$$

```
    int Sn = a1 * (pow(r, n) - 1) / (r - 1);
```

- **Formula for the sum of an infinite geometric series (when `|r| < 1`):**

$$S = \frac{a_1}{1 - r}$$

Summation / Sumatorias (Σ)

Summation (sigma notation: Σ) is used to sum a sequence of terms.

$$\sum_{i=1}^{n} i$$

```
int n = 10, sum = 0;
for (int i = 1; i <= n; ++i) {
    sum += i;
}
```

Riemann sum formula for consecutive integers:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

```
int rs = n * (n + 1) / 2;  // Direct formula instead of loop
```

Product of a secuence / productoriu / multiplicatoria (Π)

The product of a sequence (pi notation: Π) is used to multiply a sequence of terms.

$$\prod_{i=1}^{n} i$$

```
int n = 5, product = 1;
for (int i = 1; i <= n; ++i) {
    product *= i;
}
```

Factorial formula for consecutive integers:

$$\prod_{i=1}^{n} i = n!$$

```
#include <cmath>
int factorial = tgamma(n + 1);  // Using gamma function (requires <cmath>)
```

Jump Statements

- `break`: Terminates the loop or switch statement and transfers control to the statement immediately following.

```
for (int i = 1; i <= 10; ++i) {
    if (i == 5) break;
    cout << i << " ";
}
```

- `continue`: Skips the current iteration of a loop and continues with the next iteration.

```
for (int i = 1; i <= 10; ++i) {
    if (i == 5) continue;
    cout << i << " ";
}
```

- `return`: Exits a function and returns a value to the caller.

```
int add(int a, int b) {
    return a + b;
}
int result = add(3, 4);
cout << result << endl;
```

- `goto`: Transfers control to a labeled statement within the same function. (Note: can create unreadable and error-prone code, but can also solve problems with recursion).

```
int i = 1;
start:
    if (i > 5) goto end;
    cout << i << " ";
    ++i;
    goto start;
end:
```

Functions

Nesting

Nesting occurs when we nest conditionals inside one another. This leads to code that is difficult to read.

```
inline void foo() {
    if (var) {
        if (qux) {
            if (baz) {
                cout << "All conditions are true";
            } else {
                cout << "baz is false";
            }
        } else {
            cout << "qux is false";
        }
    } else {
        cout << "var is false";
    }
}
```

There are two methods to avoid nesting and become a never-nester: **inversion** and **extraction**.

1.Inversion

It consists of handling negative cases first and using return statements to exit the control flow as early as possible.

```
inline void foo() {
    if (!var) {
        cout << "var is false";
        return;
    }
    if (!qux) {
        cout << "qux is false";
        return;
    }
    if (!baz) {
        cout << "baz is false";
        return;
    }
    cout << "All conditions are true";
}
```

2.Extraction

It consists of dividing the code into smaller and more specific functions to improve readability.

```
inline void checkBaz() {
    if (!baz) {
        cout << "baz is false";
        return;
    } cout << "All conditions are true";
}
inline void checkQux() {
    if (!qux) {
        cout << "qux is false";
        return;
    } checkBaz();
}
inline void foo() {
    if (!var) {
        cout << "var is false";
        return;
    } checkQux();
}
```

Branching & Branchless

The term **branching** refers to conditionals, when the program diverges into two paths it can become slow in certain cases because the CPU tries to get ahead by preloading one of the possible functions. The branchless methodology avoids this, but it can make the function less readable.

```
inline int minorBranch(int a, int b) {
    if (a < b)
        return a;
    return b;
}
```

```
inline int minorBranchLess(int a, int b) {
    return a * (a < b) + b * (b <= a);
}
```

Lambda λ

Lambdas or lambda functions allow defining anonymous functions concisely. They are useful for creating short functions that are used in the context of another function, such as in STL algorithms.

An example:

py

```py
suma = lambda a, b: a + b
res = suma(5, 3)
```

c++

```cpp
auto suma = [](int a, int b) -> int { return a + b; };
int res = suma(5, 3);
```

Problems

- Codeforces: **4A** Watermelon
- Codeforces: **1968A** Maximize?

References

- Ceibal. (n.d.). *Tablas de verdad*. Retrieved from https://rea.ceibal.edu.uy/elp/logica-para-informatica/tablas_de_verdad.html
- code_r. (2024). *Control flow statements in Programming*. GeeksforGeeks. Retrieved from https://www.geeksforgeeks.org/control-flow-statements-in-programming/
- CodeAesthetic. (2022, January 13). *Why You Shouldn't Nest Your Code* [Video]. YouTube. https://youtu.be/CFRhGnuXG-4?si=wgfGJZ_vRLuDxTXS
- cplusplus. (n.d.). *Statements and flow control*. Retrieved from https://cplusplus.com/doc/tutorial/control/
- Creel. (2020, October 24). *Branchless Programming: Why "If" is Sloowww... and what we can do about it!* [Video]. YouTube. https://youtu.be/bVJ-mWWL7cE?si=JYBcGTo3mglW2WHn
- Low Level Learning. (2023, August 4). *why are switch statements so HECKIN fast?* [Video]. YouTube. https://youtu.be/fjUG_y5ZaL4?si=EtUvRm3a93P4KJqx
- sagar. (2023). *C++ Ternary or Conditional Operator*. GeeksforGeeks. Retrieved from https://www.geeksforgeeks.org/cpp-ternary-or-conditional-operator/
- The Cherno. (2017, August 29). *CONDITIONS and BRANCHES in C++ (if statements)* [Video]. YouTube.