PERL TO PYTHON CODE CONVERTOR

**Course: Compiler Design**

**Course Code: UE15CS351**

**Group: 14**

**Semester: 6**

**Section: ‘A’**

**Team:**

**Amogha Sekhar USN: 01FB15ECS034**

**Archit Prem USN: 01FB15ECS051**

**USEFULNESS OF PROJECT**

**PYTHON HAS THE FOLLOWING ADVANTAGES OVER PERL:**

* Python takes huge advantage over Perl when it comes to code readability. Python’s code is lot more cleaner to understand than that of Perl even when reading code after years. With indentation representing the block of code, and proper structuring, Python’s code is a lot more cleaner. Additionally in Perl, usage of ‘{‘ and ‘}’ for blocks and ‘;’ at the end of the line could make understanding the code difficult.
* Python has extensive object oriented programming support with clean and consistent syntax while OOP is Perl will add complexity to the code which will eventually make the code difficult to understand.
* There are many applications like Big Data, Infra Automation, Machine Learning and NLP for which Python is used as it has a huge support for active communities because of being Open Source.
* There are many application areas where Python is preferred and even it outperforms Perl. Like: Perl is preferred for CGI scripting but now a days [Python’s Django](https://www.tecmint.com/install-and-configure-django-web-framework-in-centos-debian-ubuntu/" \t "_blank) and web2py like web scripting languages are becoming more popular and have huge attraction from the industry.

**ALGORITHM USED**

1. If a blank or a comment line is encountered, it is printed as it is.
2. If a print statement with a newline is encountered, the matched string is stored in a variable $printInput.
   1. If $printInput contains ARGV[], its contents are outputted.
   2. If $printInput contains a variable in Perl (begins with $), the variable sign is removed and it is outputted.
   3. If $printInput contains a variable array in Perl(begins with @), the variable sign is removed and it is outputted.
   4. If there is no variable, the contents of $printInput is outputted.
3. If a print statement with no newline is encountered, the matched string is stored in $printInput.
   1. If $printInput contains a variable in Perl (begins with $), the variable sign is removed and it is outputted.
   2. If $printInput contains a variable array in Perl(begins with @), the variable sign is removed and it is outputted.
   3. If there is no variable, the contents of $printInput is outputted.
4. If last is encountered in the Perl code, which is analogous to break, break is printed as it is an equivalent of last in Python.
5. If next is encountered in the Perl code, which is analogous to continue in Python, continue is printed as it is equivalent of next in Perl.
6. If a split is encountered in the Perl code, it converts it to the format of Python. In Perl, it is SPLIT /PATTERN/, EXPR, LIMIT. In Python, it is PATTERN.split( EXPR, LIMIT).
7. If a join is encountered in the Perl code, it converts it to the format in Python. In Perl, it is JOIN EXPR, LIST. In Python it is, LIST.join(EXPR).
8. If Arithmetic expressions are encountered in the Perl code, they remain the same even in Python.
9. If ++ and – are encountered in the Perl code, the output will be var= var+1 and var=var-1 respectively as Python does not have the increment and decrement operator.
10. If a foreach is encountered( for loops in Perl) in the Perl code, it is converted to its equivalent form in Python and outputted. In Perl it is, FOREACH VAR(LIST), in Python it is FOR VAR IN RANGE().
11. If a while loop is encountered in the Perl code, it is converted to its equivalent form in Python and outputted. In Perl it is, WHILE(CONDITION){ STATEMENT(S)}. In Python it is, WHILE CONDITION: STATEMENT(S).
12. If an if is encountered in the Perl code, it is converted to its equivalent form in Python and outputted. In Perl it is IF(BOOLEAN\_EXPR){ STATEMENT(S)}. In Python it is IF BOOLEAN EXPR: STATEMENT(S). If {} are encountered, they need to be removed.
13. If an elsif is encountered in the Perl code, it is converted to its equivalent form in Python and outputted. In Perl it is ELSIF(BOOLEAN\_EXPR){ STATEMENT(S)}. In Python it is ELIF BOOLEAN EXPR: STATEMENT(S). If {} are encountered, they need to be removed.
14. If an else is encountered in the Perl code, it is converted to its equivalent form in Python and outputted. In Perl it is ELSE{ STATEMENT(S)}. In Python it is ELSE: STATEMENT(S). If {} are encountered, they need to be removed.
15. In the case an array is encountered,
    1. The declaration of the array is initially taken care of. In Perl, it is @array= (comma separated contents). In Python, it is of the form array\_var= array( comma separated list).
    2. If a push is encountered, it’s the equivalent function in Python is outputted. That is, var.push(element).
    3. If a pop is encountered, it’s equivalent function in Python is outputted. That is, var.pop.
    4. If an unshift is encountered, it’s equivalent function in Python is outputted. That is, var.unshift(element).
    5. If a shift is encountered, it’s equivalent function in Python is outputted. That is, var.shift.
16. If none of the above cases match(1-15), the line is outputted.
17. Repeat steps 1 to 16 for each line in the input file.

**REGEX USED AND EXPLANATIONS**

**\s** = space character = \t\n\r in C

**^** = Beginning of string

**(.\*)** = Any number of any characters except newline

**$line =~ /^\s\*#/ || $line =~ /^\s\*$/** : This regular expression matches any number of space characters followed by # or any number of space characters. If there are only spaces in a line, it is a blank line. Matches all blank lines and comments.

**$line =~ /^\s\*print\s\*"(.\*)\\n"[\s;]\*$/** : This regular expression matches a print statement followed by \n followed by space followed by ; .

**$printInput =~ /ARGV\[(.\*)\]$/** : This regular expression matches variable number of arguments being provided on the command line.

**$printInput =~ /^(.\*)\s\*\$(.\*)\*$/** : This regular expression matches a line with exactly one $variable. Anything before or after '$' can be any character.

**$printInput =~ s/\$//** : This regular expression is of the form s/pattern/replace. In this case, $ is replaced with nothing. It basically removes $ from printInput.

**$printInput =~ /^(.\*)\s\*\@(.\*)\*$/** : This regular expression matches a line with exactly one @variable. Anything before or after '@' can be any character.

**$printInput =~ s/\@//** : This regular expression is to remove '@' from printInput.

**$line =~ /^\s\*print\s\*"(.\*)"[\s;]\*$/** : This regular expression matches a print statement followed by ; .

**$printInput =~ /^\s\*print\s\*\$\\_\s\*;$/** : This regular expression matches a line with exactly one default variable. $\_ is a special variable that contains the default input. If print is used in a loop, the value of print will be stored in $\_ for each iteration of the loop.

**$line =~ /^\s\*last;$/** : This regular expression matches a line that contains 'last'. Last is a keyword that causes the iteration of the loop in which it is called, to be the last iteration. It's analogous to break in C.

**$line =~ /^\s\*next;$/** : This regular expression matches a line that contains 'next'. It is analogous to continue.

**$line =~ /^\s\*(.\*)\s\*=\s\*split\(\/(.\*)\/,\s\*\$(.\*)\)\s\*;/ :** This regular expression matches a line in which the split function is used. Split function is used in this manner : split(/:/, "1:2:3:4:5"). This RegEx matches a line that looks like the representation above.

**$line =~ /^\s\*(.\*)\s\*=\s\*join\(\'(.\*)\'\,\s\*(.\*)\)\s\*;$/** : This regular expression matches a line in which join function is used. Join function is used like : Join Expr, List. Combines elements of List and merges it to form a string, Expr..

**$line =~ /^\s\*while\s\*(.\*)\<\>(.\*)\s\*(.\*)\s\*$/**: This regular expression matches a while statement in Perl.

**$line =~ /^\s\*[^\s]\*\s\*=(.\*);$/**: This regular expression matches an arithmetic expression in Perl.

**$line =~ /^\s\*(.\*)\s\*\+\+(.\*);$/**: This regular expression matches the increment (++) operator in Perl.

**$line =~ /^\s\*(.\*)\s\*\-\-(.\*);$/**: This regular expression matches the decrement (--) operator in Perl.

**$line =~ /^\s\*foreach\s\*\$(.\*)\s\*\((.\*)\)\s\*{\s\*$/**: This regular expression is used to match arrays of the form foreach var(list) in Perl.

**$line =~ /^\s\*(.\*)\s\*while\s\*\((.\*)\)(.\*)\s\*$/**: This regular expression matches the while loop in Perl.

**$line =~ /^\s\*(.\*)\s\*if\s\*\((.\*)\)(.\*)\s\*$/**: This regular expression matches the if statement in Perl.

**$line =~ /^\s\*(.\*)\s\*elsif\s\*\((.\*)\)(.\*)\s\*$/**: This regular expression matches the elsif statement in Perl.

**$line =~ /^\s\*(.\*)\s\*else\s\*(.\*)\s\*$/**: This regular expression matches the else statement in Perl.

**$line =~ /^\s\*}\s\*$/**: This regular expression matches the ‘}’ in Perl.

**$line =~ /^\s\*(.\*)\s\*\@(.\*)\s\*(.\*)\s\*;$/**: This regular expression matches an array in Perl.

**$line =~ /^\s\*push\s\*\@(.\*)\,\s\*(.\*)\s\*;$/**: This regular expression matches the function push in Perl. This function has a definition push @ARRAY, LIST. It pushes the values in LIST onto the end of the list @ARRAY.

**$line =~ /^\s\*pop\s\*\@(.\*);$/**: This regular expression matches the function pop in Perl. This function has a definition POP(@ARRAY). It is used to remove the last element from an array.

**$line =~ /^\s\*unshift\s\*\@(.\*)\,\s\*(.\*)\s\*;$/**: This regular expression matches the function unshift in Perl. This function has the definition UNSHIFT @ARRAY, LIST. It places the elements in LIST in order at the beginning of the @ARRAY.

**$line =~ /^\s\*shift\s\*\@(.\*);$/**: This regular expression matches the function shift in Perl. This function has a definition SHIFT(@ARRAY). It removes the first element of the array.

**READ ME FILE**

Perl2Python:

A tool to attempt conversion of Perl scripts to Python 2.7.

Use:

Run the script on the command line. Argument 1 needs to be the file to translate (multiple files will be ignored).

Output will be written to stdout

If a line is unable to be translated, it will be output as a comment. You'll need to check this manually, as no warnings will be thrown and this can cause serious breakage.

**TEST CASES**

**1. Input:**

$a = 12;

$b = 2;

$c = $a + $b \* $b;

print "$c\n"

**Output:**

a = 12

b = 2

c = a + b \* b

print c

**2. Input:**

$a = 12;

$b = 2;

$c = $a + $b / $b;

if ($c != 0 && $a < 0) {

print "Not right. No value for you.\n";

} elsif ($a > 20) {

print "This program should correct my incorrect syntax here.";

} else {

print "$c"

}

**Output:**

a = 12

b = 2

c = a + b // b

if c != 0 and a < 0:

print "Not right. No value for you."

elif a > 20:

sys.stdout.write("This program should correct my incorrect syntax here.")

else:

sys.stdout.write(c)

**3. Input:**

$answer = 0;

while ($answer < 36) {

$answer = $answer + 7;

if ($answer == 9) {

$answer ++;

last;

} else {

next;

}

}

print "$answer\n";

**Output:**

answer = 0

while answer < 36:

answer = answer + 7

if answer == 9:

answer +=1

break

else:

continue

print answer

**4. Input:**

@a = (6, 7, 8, 9);

push @a, 9;

pop @a;

unshift @a, 5;

shift @a;

**Output:**

push a, 9

pop a

unshift a, 5

shift a

**5. Input:**

$n = 1;

while ($n <= 10) {

$total = 0;

$j = 1;

while ($j <= $n) {

$i = 1;

while ($i <= $j) {

$total = $total + $i;

$i = $i + 1;

}

$j = $j + 1;

}

print "$total\n";

$n = $n + 1;

}

**Output:**

n = 1

while n <= 10:

total = 0

j = 1

while j <= n:

i = 1

while i <= j:

total = total + i

i = i + 1

j = j + 1

print total

n= n+1