Η Γλώσσα Προγραμματισμού $\mathfrak{C}ute\mathcal{P}y$

Γεώργιος Μανής

Πανεπιστήμιο Ιωαννίνων
Πολυτεχνική Σχολή
Τμήμα Μηχανικών Η/Υ και Πληροφορικής

Φεβρουάριος 2023

Η $\mathfrak{C}ute\mathcal{P}y$ είναι μια μικρή, εκπαιδευτική, γλώσσα προγραμματισμού. Θυμίζει τη γλώσσα Python, από την οποία αντλεί ιδέες και δομές, αλλά είναι αρκετά πιο μικρή, τόσο στις υποστηριζόμενες δομές, όσο φυσικά και σε προγραμματιστικές δυνατότητες. Τα προγράμματα $\mathfrak{C}ute\mathcal{P}y$ είναι γραμμένα με τέτοιο τρόπο, ώστε να μπορούν να τρέξουν και από έναν διερμηνέα Python.

Παρόλο που οι προγραμματιστικές της δυνατότητες είναι μικρές, η εκπαιδευτική αυτή γλώσσα προγραμματισμού περιέχει πλούσια στοιχεία και η κατασκευή του μεταγλωττιστή της έχει να παρουσιάσει αρκετό ενδιαφέρον, αφού περιέχονται σε αυτήν δημοφιλείς δομές, όπως οι `while` και `if-else`. Επίσης, υποστηρίζει συναρτήσεις, μετάδοση παραμέτρων με τιμή και αναδρομικές κλήσεις, αλλά και επιτρέπει φώλιασμα στη δήλωση συναρτήσεων.

Από την άλλη όμως πλευρά, η $\mathfrak{C}ute\mathcal{P}y$ δεν προσφέρει βασικά προγραμματιστικά εργαλεία όπως η δομή *for* ή τύπους δεδομένων όπως οι πραγματικοί αριθμοί και οι συμβολοσειρές ή οι πίνακες. Οι παραλήψεις αυτές έχουν γίνει για εκπαιδευτικούς λόγους, ώστε να απλουστευτεί η διαδικασία κατασκευής του μεταγλωττιστή. Είναι μία απλοποίηση που έχει να κάνει μόνο με τη μείωση των γραμμών κώδικα που πρέπει να γραφεί και όχι με τη δυσκολία της κατασκευής ή την εκπαιδευτική αξία της διαδικασίας ανάπτυξης.

Παρακάτω, θα οριστεί η γλώσσα $\mathfrak{C}ute\mathcal{P}y$. Θα παρουσιαστεί η δομή ενός προγράμματος σε γλώσσα $\mathfrak{C}ute\mathcal{P}y$, οι κανόνες φωλιάσματος συναρτήσεων, οι κανόνες εμβέλειας και ο τρόπος περάσματος παραμέτρων.

Η ανάπτυξη του μεταγλωττιστή $\mathfrak{C}ute\mathcal{P}y$ θα γίνει χωρίς τη χρήση οποιονδήποτε εργαλείων ανάπτυξης, αλλά μόνο με τη χρήση μιας γλώσσας προγραμματισμού, της Python. Ο μεταγλωττιστής θα είναι πλήρως λειτουργικός και θα παράγει, ως τελική γλώσσα, τη γλώσσα μηχανής του επεξεργαστή RISC-V. Χρησιμοποιώντας έναν εξομοιωτή του επεξεργαστή RISC-V θα είναι δυνατόν να τρέξουμε τη γλώσσα μηχανής που θα δημιουργεί ο μεταγλωττιστής. Η επιλογή της γλώσσας μηχανής του RISC-V ως γλώσσα στόχο, έχει να κάνει με εκπαιδευτικούς λόγους, αφού η επιλογή ενός εμπορικού επεξεργαστή δεν έχει να προσθέσει κάτι εκπαιδευτικά.

Τα αρχεία της $\mathfrak{C}ute\mathcal{P}y$ έχουν κατάληξη `.cpy`

Λεκτικές μονάδες:

Το αλφάβητο της $\mathfrak{C}ute\mathcal{P}y$ αποτελείται από:

- τα μικρά και κεφαλαία γράμματα της λατινικής αλφαβήτου (`A,…,Z` και `a,…,z`),
- τα αριθμητικά ψηφία (`0,…,9`),
- την κάτω παύλα (_),
- τα σύμβολα των αριθμητικών πράξεων (`+, -, *, //`),
- τους τελεστές συσχέτισης (`<, >, !=, <=, >=, ==`)
- το σύμβολο ανάθεσης (`=`)
- τους διαχωριστές (`;, “,”, :`)
- τα σύμβολα ομαδοποίησης (`[, ], (, ) , #{ , #}`)
- και διαχωρισμού σχολίων (`#$`)

Τα σύμβολα `[` , `]` χρησιμοποιούνται στις λογικές παραστάσεις όπως τα σύμβολα `(` , `)` στις αριθμητικές παραστάσεις.

Οι δεσμευμένες λέξεις της γλώσσας δεν μπορούν να χρησιμοποιηθούν ως μεταβλητές.

Οι σταθερές της γλώσσας είναι ακέραιες σταθερές που αποτελούνται από προαιρετικό πρόσημο και από μία ακολουθία αριθμητικών ψηφίων. Οι ακέραιες σταθερές πρέπει να έχουν τιμές από $-(2^{32}-1)$ έως $2^{32}-1$.

Τα αναγνωριστικά της γλώσσας είναι συμβολοσειρές που αποτελούνται από γράμματα, ψηφία και κάτω παύλες, αρχίζοντας όμως από γράμμα. Ο μεταγλωττιστής λαμβάνει υπόψη του μόνο τα τριάντα πρώτα γράμματα. Αναγνωριστικά με περισσότερους από 30 χαρακτήρες θεωρούνται λανθασμένα.

Οι λευκοί χαρακτήρες (tab, space, return) αγνοούνται και μπορούν να χρησιμοποιηθούν με οποιονδήποτε τρόπο χωρίς να επηρεάζεται η λειτουργία του μεταγλωττιστή, αρκεί βέβαια, να μην βρίσκονται μέσα σε δεσμευμένες λέξεις, αναγνωριστικά, αριθμητικές σταθερές.

Το ίδιο ισχύει και για τα σχόλια, τα οποία πρέπει να βρίσκονται μέσα σε σύμβολα `#$`.

Μορφή προγράμματος:

Κάθε πρόγραμμα αποτελείται από δύο τμήματα. Στο πρώτο τμήμα δηλώνονται οι *κύριες συναρτήσεις*. Πρόκειται για συναρτήσεις χωρίς παραμέτρους οι οποίες δεν επιστρέφουν αποτέλεσμα. Ξεκινούν με το διακριτικό `main_`. Δηλαδή, νόμιμα ονόματα για μία κύρια συνάρτηση μπορεί να είναι τα ακόλουθα: `main_function, main_finonacci, main_foo`.

Κάθε κύρια συνάρτηση μπορεί να περιέχει μία ή περισσότερες φωλιασμένες *τοπικές συναρτήσεις*. Επιτρέπεται φωλιασμός τοπικών συναρτήσεων, δεν επιτρέπεται φωλιασμός ανάμεσα σε κύριες συναρτήσεις. Δηλαδή, μία κύρια συνάρτηση δεν μπορεί να περιέχει μέσα της μία κύρια συνάρτηση. Επίσης, μία κύρια συνάρτηση δεν μπορεί να καλέσει μία κύρια συνάρτηση.

Η δομή μίας κύριας συνάρτησης φαίνεται παρακάτω:

```
def main_function()
#{
    declarations
    local_functions
    statements
#}
```

Κάθε κύρια συνάρτηση μπορεί να δηλώσει μέσα της και να καλέσει τοπικές συναρτήσεις, σύμφωνα με τους κανόνες εμβέλειας. Έτσι, κάθε `functions` στην παραπάνω περιγραφή μέσα σε μία κύρια συνάρτηση μπορεί να περιέχει μία ή περισσότερες τοπικές συναρτήσεις της μορφής:

```
local_functions  --> ( local_function )*
```

δηλαδή:

```
def local_function(formal_parameters)
#{
    declarations
    local_functions
    statements
#}
```

Η αναδρομικότητα στη δήλωση των τοπικών είναι προφανής, σε αντίθεση με τις κύριες συναρτήσεις.

Τύποι και δηλώσεις μεταβλητών:

Ο μοναδικός τύπος δεδομένων που υποστηρίζει η $\mathfrak{C}ute\mathcal{P}y$ είναι οι ακέραιοι αριθμοί. Η δήλωση γίνεται με την εντολή `#declare`. Ακολουθούν τα ονόματα των αναγνωριστικών χωρίς καμία άλλη δήλωση, αφού γνωρίζουμε ότι πρόκειται για ακέραιες μεταβλητές και χωρίς να είναι αναγκαίο να βρίσκονται στην ίδια γραμμή. Οι μεταβλητές χωρίζονται μεταξύ τους με κόμματα. ~~Το τέλος της~~

δήλωσης αναγνωρίζεται με το ελληνικό ερωτηματικό. Επιτρέπεται να έχουμε περισσότερες των μία συνεχόμενες χρήσεις της `#declare`.

Τελεστές και εκφράσεις:

Η προτεραιότητα των τελεστών από τη μεγαλύτερη στη μικρότερη είναι:

- Πολλαπλασιαστικοί: `*, //`
- Προσθετικοί: `+, -`
- Σχεσιακοί: `==, <, >, !=, <=, >=`
- Λογικό `not`
- Λογικό `and`
- Λογικό `or`

Δομές της γλώσσας:

Οι δομές της γλώσσας χωρίζονται σε δύο κατηγορίες: τις απλές εντολές και τις δομημένες. Οι απλές εντολές τελειώνουν με ένα ελληνικό ερωτηματικό. Στις δομημένες δεν απαιτείται να τελειώνουν με ελληνικό ερωτηματικό. Απλές εντολές θεωρούμε τις εντολές εκχώρησης, επιστροφής τιμής συνάρτηση, εισόδου και εξόδου δεδομένων. Δομημένες εντολές στην $\mathfrak{C}ute\mathcal{P}y$ θεωρούμε τις εντολές απόφασης και επανάληψης. Παρακάτω θα δούμε μία-μία τις δομές της γλώσσας.

Εκχώρηση:

Χρησιμοποιείται για την ανάθεση τιμής σε μιας μεταβλητής.

Σύνταξη:

```
var_name = expression;
```

όπου *expression* είναι μία μαθηματική έκφραση.

Απόφαση if:

Η εντολή απόφασης `if` εκτιμά εάν ισχύει η συνθήκη *condition* και εάν πράγματι ισχύει, τότε εκτελούνται οι εντολές $statements_1$ που το ακολουθούν. Το `else` δεν αποτελεί υποχρεωτικό τμήμα της εντολής και γι' αυτό βρίσκεται μέσα

σε αγκύλες. Οι εντολές *statements*$_2$ που ακολουθούν το `else` εκτελούνται εάν η συνθήκη *condition* δεν ισχύει.

Τα *statements* μπορεί να είναι μία απλή ή μία δομημένη εντολή. Μπορεί επίσης να είναι μία σειρά από τέτοιες εντολές, οι οποίες είναι κλεισμένες ανάμεσα στα σύμβολα `#{` και `#}`. Η σύνταξη αυτή είναι η συνήθης σύνταξη τέτοιων δομών στις δημοφιλείς γλώσσες προγραμματισμού.

Σύνταξη:

```
if (condition):
    statements_1
[ else:
    statements_2 ]
```

Επανάληψη while:

Η εντολή επανάληψης `while` επαναλαμβάνει τις εντολές *statements*, όσο η συνθήκη *condition* ισχύει. Αν την πρώτη φορά που θα αποτιμηθεί η *condition*, το αποτέλεσμα της αποτίμησης είναι ψευδές, τότε οι *statements* δεν εκτελούνται ποτέ.

Σύνταξη:

```
while (condition):
    statements
```

Το *statements* είναι το ίδιο με αυτό που περιγράφηκε στη δομή `if`. Μπορεί να είναι μία εντολή ή μία σειρά από εντολές, οι οποίες στη δεύτερη περίπτωση, είναι κλεισμένες ανάμεσα στα σύμβολα `#{` και `#}`.

Επιστροφή τιμής συνάρτησης:

Χρησιμοποιείται μέσα σε τοπικές συναρτήσεις για να επιστραφεί το αποτέλεσμα της συνάρτησης, το οποίο είναι το αποτέλεσμα της αποτίμησης του *expression*.

Σύνταξη:

```
return (expression);
```

Έξοδος δεδομένων:

Εμφανίζει στην οθόνη το αποτέλεσμα της αποτίμησης του *expression*.

Σύνταξη:

```
print (expression);
```

Είσοδος δεδομένων:

Ζητάει από τον χρήστη να δώσει μία τιμή μέσα από το πληκτρολόγιο. Η τιμή που θα δώσει θα μεταφερθεί στην μεταβλητή *var*, μέσω της εκχώρησης.

Σύνταξη:

```
var = int(input());
```

Συναρτήσεις:

Η $\mathfrak{C}ute\mathcal{P}y$ υποστηρίζει κύριες και τοπικές συναρτήσεις. Η σύνταξη της δήλωσης των τοπικών συναρτήσεων είναι:

```
def function(formal_parameters)
#{
    declarations
    functions
    statements
#}
```

Η *formal_parameters* είναι η λίστα των τυπικών παραμέτρων. Οι τοπικές συναρτήσεις μπορούν να φωλιάσουν η μία μέσα στην άλλη και οι κανόνες εμβέλειας είναι όπως αυτοί της PASCAL. Η επιστροφή της τιμής μιας τοπικής συνάρτησης γίνεται με την εντολή `return`.

Η κλήση μιας τοπικής συνάρτησης, γίνεται από τις αριθμητικές παραστάσεις σαν τελούμενο. π.χ. Έτσι οι ακόλουθες κλήσεις για την τοπική συνάρτηση `foo`είναι ορθές:

```
x = foo(...);
y = foo(...)//2 + 4;
```

Μετάδοση παραμέτρων:

Η $\mathfrak{C}ute\mathcal{P}y$ υποστηρίζει μετάδοση παραμέτρων *με τιμή*, όπως η Python. Πιθανές αλλαγές στην τιμή της κατά την εκτέλεση της τοπικής συνάρτηση δεν επιστρέφονται στη συνάρτηση που την κάλεσε.

Κανόνες εμβέλειας:

Καθολικές ονομάζονται οι μεταβλητές που δηλώνονται στις κυρίως συναρτήσεις και είναι προσβάσιμες από όλες τις τοπικές συναρτήσεις της κύριας συνάρτησης. *Τοπικές* είναι οι μεταβλητές που δηλώνονται σε μία τοπική συνάρτηση και είναι προσβάσιμες μόνο μέσα από τη συγκεκριμένη συνάρτηση. Κάθε συνάρτηση, εκτός των τοπικών μεταβλητών, των παραμέτρων και των καθολικών μεταβλητών, έχει επίσης πρόσβαση και στις μεταβλητές που έχουν δηλωθεί σε συναρτήσεις προγόνους ή και σαν παράμετροι αυτών.

Ισχύει ο δημοφιλής κανόνας ότι, αν δύο (ή περισσότερες) μεταβλητές ή παράμετροι έχουν το ίδιο όνομα και έχουν δηλωθεί σε διαφορετικό επίπεδο φωλιάσματος, τότε οι τοπικές μεταβλητές και παράμετροι υπερκαλύπτουν τις μεταβλητές και παραμέτρους των προγόνων, οι οποίες με τη σειρά τους υπερκαλύπτουν τις καθολικές μεταβλητές.

Μία συνάρτηση έχει δικαίωμα να καλέσει τον εαυτό της και όποια συνάρτηση βρίσκεται στο ίδιο επίπεδο φωλιάσματος με αυτήν, αρκεί η δήλωσή της συνάρτησης που καλείται να προηγείται στον κώδικα της συνάρτησης που την καλεί.

Κλήση κυρίων συναρτήσεων:

Οι κύριες συναρτήσεις καλούνται μόνο από το κυρίως πρόγραμμα. Δεν επιστρέφουν τιμή, δεν δέχονται παραμέτρους, δεν καλούν άλλες κύριες συναρτήσεις, παρά μόνο εκτελούν τον δικό τους κώδικα και καλούν τις δικές τους απλές συναρτήσεις.

Το κυρίως πρόγραμμα δηλώνεται μετά το τέλος των κυρίων συναρτήσεων. Ο τρόπος δήλωσης δανείζεται από την Python, μέσα από την:

```
if __name__ == "__main__":
```

Ένα παράδειγμα ακολουθεί:

```
if __name__ == "__main__":
    main_function1()
    main_function2()
    main_function3()
```

Η γραμματική της $\mathfrak{C}ute\mathcal{P}y$:

Ακολουθεί η γραμματική της $\mathfrak{C}ute\mathcal{P}y$ η οποία δίνει και την ακριβή περιγραφή της γλώσσας:

```
startRule
        :   def_main_part
            call_main_part
        ;

def_main_part
        :   ( def_main_function )+
        ;

def_main_function
        :   'def' ID '(' ')' ':'
            '#{'
                    declarations
                    ( def_function )*
                    statements
            '#}'
        ;

def_function
        :   'def' ID '(' id_list ')' ':'
            '#{'
                    declarations
                    ( def_function )*
                    statements
            '#}'
        ;

declarations
        :   ( declaration_line )*
        ;
```

```
declaration_line
        :   '#declare' id_list
        ;

statement
        :   simple_statement
        |   structured_statement
        ;

statements
        :   statement+
        ;

simple_statement
        :   assignment_stat
        |   print_stat
        |   return_stat
        ;

structured_statement
        :   if_stat
        |   while_stat
        ;

assignment_stat
        :   ID '='
            (   expression ';'
            |   'int' '(' 'input' '(' ')' ')' ';'
            )
        ;

print_stat
        :   'print' '(' expression ')' ';'
        ;

return_stat
        :   'return' '(' expression ')' ';'
        ;
```

```
if_stat
        :   'if' '(' condition ')' ':'
                (   statement
                |   '#{' statements '#}'
                )
            (   'else' ':'
                (   statement
                |   '#{' statements '#}'
                )
            )?
        ;

while_stat
        :   'while' '(' condition ')' ':'
                (   statement
                |   '#{' statements '#}'
                )
        ;

id_list
        :     ID ( ',' ID )*
        |
        ;

expression
        :   optional_sign term
            ( ADD_OP term )*
        ;

term
        :   factor
            ( MUL_OP factor )*
        ;

factor
        :   INTEGER
        |   '(' expression ')'
        |    ID idtail
        ;
```

```
idtail
        : '(' actual_par_list ')'
        |
        ;

actual_par_list
        :     expression (  ',' expression )*
        |
        ;

optional_sign
        :    ADD_OP
        |
        ;

condition
        :   bool_term  ( 'or' bool_term )*
        ;

bool_term
        :   bool_factor ( 'and' bool_factor )*
        ;

bool_factor
        :   'not' '[' condition ']'
        |   '[' condition ']'
        |   expression REL_OP expression
        ;

call_main_part
        :
        'if' '__name__' '==' '"__main__"' ':'
            (  main_function_call )+
        ;

main_function_call
        :   ID '(' ')' ';'
        ;
```

Παράδειγμα σε γλώσσα $\mathfrak{C}ute\mathcal{P}y$:

Το παρακάτω παράδειγμα έχει γραφεί σε γλώσσα $\mathfrak{C}ute\mathcal{P}y$. Μπορεί να εκτελεστεί και από έναν διερμηνέα Python.

```
def main_factorial():
#{
    #$ declarations #$
    #declare x
    #declare i,fact

    #$ body of main_factorial #$
    x = int(input());
    fact = 1;
    i = 1;
    while (i<=x):
    #{
        fact = fact * i;
        i = i + 1;
    #}
    print(fact);
#}



def main_fibonacci():
#{e
    #declare x

    def fibonacci(x):
    #{
        if (x<=1):
            return(x);
        else:
            return (fibonacci(x-1)+fibonacci(x-2));
    #}

    x = int(input());
    print(fibonacci(x));
#}
```

```
def main_countdigits():
#{
    #declare x, count

    x = int(input());
    count = 0;
    while (x>0):
    #{
        x = x // 10;
        count = count + 1;
    #}
    print(count);
#}



def main_primes():
#{
    #declare i

    def isPrime(x):
    #{
        #declare ibasic

        def divides(x,y):
        #{
            if (y == (y//x) * x):
                return (1);
            else:
                return (0);
        #}

        i = 2;
        while (i<x):
        #{
            if (divides(i,x)==1):
                return (0);
            i = i + 1;
        #}
        return (1);
    #}
```

```
    #$ body of main_primes #$
    i = 2;

    while (i<=30):
    #{
        if (isPrime(i)==1):
            print(i);
        i = i + 1;
    #}

#}


if __name__ == "__main__":
    #$  call of main functions #$
    main_factorial();
    main_fibonacci();
    main_countdigits();
    main_primes();
```