EduPi

Federico Guida, Giovanni Raccuglia, Alessio Princiotta

July 11, 2020

Progetto Linguaggi e Traduttori 2019/2020
Docente: Antonio Chella

Contents

1 Introduzione

EDUPI (EDUcational raspberryPI) è un linguaggio di programmazione il cui
scopo è quello di semplificare l'apprendimento delle basi di programmazione. L'obiettivo è quello di rendere i paradigmi della programmazione più accessibili ai principianti semplificando le procedure di utilizzo dei protocolli e di impiego delle periferiche.
Inoltre il linguaggio si focalizza sullo sviluppo di codice eseguibile su una specifica macchina target Raspberry Pi comprendendo una serie di versioni di quest'ultima (dalla versione Pi 1 alla più recente Pi 4B). La relazione spazierà su varie tematiche partendo dalla sezione 2 che si occuperà di illustrare il contesto dove si colloca il linguaggio, continuando con la sezione 3 che descriverà il linguaggio, con la sezione 4 illustreremo le peculiarità del nostro linguaggio, infine concluderemo con la sezione 7 dove faremo delle considerazioni sul lavoro svolto.

2 Stato dell'arte

Il **Raspberry Pi** è una macchina utilizzata per svariati scopi che spaziano dal General-Pourpose a scopi più specifici come la programmazione di Sistemi Embedded. Il linguaggio EduPi permette attraverso poche righe di codice l'utilizzo di determinate periferiche anche utilizzando protocolli complessi. Un punto di forza del linguaggio proposto permette essendo interattivo il testing delle funzionalità delle periferiche e del comportamento della macchina target. Il Raspberry Pi è stato scelto come macchina target in quanto è largamente utilizzata a scopo educativo essendo un dispositivo facile da utilizzare e anche abbastanza resistente. La famiglia *Raspberry Pi Single Board Computer (SBC)* ha guadagnato popolarità in diverse aree, mentre l'educazione rimane il motore fondamentale dietro il design. I kit a basso costo sono forniti, in particolare per l'istruzione, dalla Fondazione Raspberry Pi in collaborazione con Google. Questi kit producono risultati educativi ottimali e un'esperienza arricchita. Sono disponibili numerose risorse Web in continua evoluzione, nonché implementazioni hardware crowdfunded [1]. EduPi, grazie alla sua semplicità, risulta ideale nell'utilizzo in ambiente scolastico permettendo ai professori di insegnare ai propri alunni in maniera semplice alcuni aspetti della programmazione e dell'elettronica già dai primi anni del percorso di studi.

3 Descrizione del progetto

EduPi è un linguaggio creato prendendo spunto dalla sintassi dei linguaggi più utilizzati nell'ambito della programmazione. I linguaggi che hanno influenzato lo sviluppo di EduPi sono stati: *Python e C*. La scelta è stata fatta per affiancare alla programmazione interattiva l'utilizzo di una sintassi appartenente al bagaglio culturale dello studente. Lo scopo è quello di mantenere l'attenzione sull'utilizzo del dispositivo target e delle periferiche.
Una delle caratteristiche principali di EduPi è la presenza del tipo *peripheral*, un tipo composto che permette l'utilizzo facilitato da parte dello studente di tutti i dispositivi messi a disposizione in aula. Oltre l'implementazione di questo tipo composto, sono stati introdotti una serie di tipi sia primitivi che composti che mantengono l'utilizzo dei paradigmi della programmazione tradizionale. I tipi che sono stati introdotti sono *integer, real, string, list*. Abbiamo scelto di non introdurre volutamente il tipo *array*, in quanto abbiamo reso disponibili una serie di funzionalità che permettono l'utilizzo delle liste in maniera più flessibile e immediata rispetto al tipo *array*. Inoltre, per facilitare l'accesso agli elementi della lista è stato introdotto il costrutto for each, con l'utilizzo di una sintassi più semplice rispetto al tradizionale for permettendo di iterare in maniera efficace gli elementi di una lista. Per quanto riguarda le stringhe abbiamo deciso di non introdurre un tipo primitivo *char* poichè abbiamo considerato il tipo *string* più semplice da utilizzare per gli alunni alle prime armi con la programmazione, e per facilitare la loro manipolazione essendo immutabili abbiamo introdotto una serie di *builtin*.

3.1 Attributi del linguaggio

Per quanto riguarda gli attributi del nostro linguaggio abbiamo scelto di focalizzarci su alcuni specifici.

- `Leggibilità:` EduPi possiede buona leggibilità, abbiamo deciso di basarci sullo stile di programmazione dei linguaggi più utilizzati, garantendo una buona facilità di manutenzione.
- `Chiarezza:` EduPi risulta essere un linguaggio chiaro e semplice, possiede soltanto i costrutti di base che servono per lo scopo del linguaggio, garantendo così il suo utilizzo e il suo apprendimento in tempi brevi.
- `Ortogonalità:` Nello sviluppo di EduPi abbiamo cercato di utilizzare i costrutti definiti per tutti i tipi a disposizione.Ad esempio è possibile sommare un *Integer* con un *String* valutando quest'ultimo come un numero.
- `Naturalezza di applicazione:` EduPi permette la scrittura di applicazioni per il quale è stato progettato nel modo più semplice possibile.
- `Facilità della verifica della correttezza:` EduPi ha un alto controllo degli errori permettendo così di verificare se il programma sta facendo quello per cui è stato progettato.
- `Ambiente di programmazione:` Per facilitare l'utilizzo e la scrittura di programmi in EduPi è stata fatta un estensione per Visual Studio Code che permette il syntax highlighting e l'interazione con alcune funzionalità tipiche dei linguaggi di programmazione tradizionali. *E' possibile scaricare l'estensione semplicemente cercando* `edupi` *nello store Microsoft.*
- `Costo di utilizzo:` EduPi è un linguaggio che verrà utilizzato su una macchina con risorse limitate, quindi il team ha deciso di ottimizzare il più possibile il linguaggio. Siamo partiti da un analisi della memory leak, grazie all'utilizzo di *valgrind* framework programmabile per la creazione di strumenti di supervisione di programmi come rilevatori di bug e profiler. Esegue programmi supervisionati utilizzando la traduzione binaria dinamica, dandogli il controllo totale su ogni parte senza richiedere il codice sorgente e senza la necessità di ricompilare o ricollegare prima dell'esecuzione. [2].Infine abbiamo fatto dei test che verranno riportati nelle sezioni successive per il calcolo computazionale delle istruzioni eseguite comparandole con i linguaggi più utilizzati dell'ultimo decennio.

3.2 Scelte progettuali

Durante la progettazione di EduPi il team ha effettuato una serie di scelte che hanno permesso di soddisfare gli attribuiti definiti nella sezione 3.1. Abbiamo deciso di separare i numeri *interi* da quelli *reali* per evitare errori nella precisione dei risultati. Per quanto riguarda le funzioni per facilitarne l'utilizzo non abbiamo previsto tipo di ritorno, garantendo così di definire sia funzioni void, oppure funzioni con ritorno utilizzando la parola chiave *return* seguita dalla variabile/valore da ritornare. I parametri delle funzioni sono opportunamente gestisti garantendo la loro esistenza solo all'interno della funzione. Il team in questa prima versione di EduPi non gestisce lo scope e utilizza variabili globali, quindi per evitare problemi durante l'esecuzione del programma si consiglia di definire le variabili *con nomi diversi* comprese quelle all'interno dei parametri per definire una funzione. Una delle caratteristiche fondamentali delle variabili in EduPi è la loro versatilità, se una variabile definita con un determinato tipo e nome non è più utile all'interno del programma è possibile ri-definirla con lo stesso nome affiancandogli un tipo diverso. Il team ha prestato molta attenzione all'assegnazione delle variabili, sono stati fatti tutti gli opportuni controlli per evitare che un valore primitivo o composto sia assegnato ad una variabile con un tipo differente. Quindi anche se i parametri delle funzioni sono senza tipo al momento della dichiarazione la loro manipolazione durante la chiamata a funzione viene opportunamente gestita. EduPi si presenta come un linguaggio in cui è possibile effettuare numerose operazioni, il team ha cercato di permettere l'utilizzo di quest'ultime a più tipi possibili, ove non è possibile utilizzare un determinato tipo per un operazione verrà eseguito un messaggio di errore per avvisare l'utilizzatore del programma della non compatibilità del tipo delle variabili utilizzate nella specifica operazione. Inoltre il team permette tramite la funzione *type(var)* di identificare il tipo di una variabile e quindi da la possibilità anche al programmatore di gestire operazioni definite da quest'ultimo in base ai tipi. Il team ha introdotto una serie di funzioni di supporto come time(), rand(), sqrt() etc.. che permettono così la definizione di comportamenti complessi da parte del programmatore. Grazie all'utilizzo della funzione time() ad esempio il programmatore potrebbe definire determinati comportamenti del Raspberry Pi in base

al tempo restituito della funzione, per facilità di utilizzo di questa funzione il team ha deciso che restituisca una stringa che rappresenti il tempo. Uno dei punti di forza di EduPi è il tipo *peripheral* che permette di inserire una descrizione della periferica e una serie di comportamenti definiti da funzioni pre-assemblate da parte di un programmatore più esperto.Il tipo *peripheral* è un tipo statico composto secondo la nostra scelta progettuale in quanto le sue funzionalità che definisco i comportamenti delle periferiche non possono essere modificati aggiunti o rimossi dopo la sua inizializzazione. Per semplificare la definizione di questi comportamenti il team ha previsto che EduPi sia un linguaggio che possa essere eseguito accompagnato da ulteriori parametri. EduPi può essere eseguito in tre modalità differenti:

1. **Modalità Esperta** - Modalità senza controlli interattiva.
2. **Modalità Controllata** - Modalità interattiva costituita da controlli che permettono il giusto utilizzo delle funzioni dedicate al Raspberry Pi.
3. **Modalità File** - Modalità senza controlli che permette il passaggio di un file come parametro per l'inclusione di tutti i comportamenti definiti da un programmatore esperto del linguaggio, l'esecuzione di EduPi terminerà alla fine della lettura del file.
4. **Modalità Sicura con File** - Modalità controllata che permette il passaggio di un file come parametro per l'inclusione di tutti i comportamenti definiti da un programmatore esperto del linguaggio, l'esecuzione di EduPi continuerà alla fine della lettura del file, permettendo ai programmatori meno esperti di utilizzare le funzioni definite.

```
1. sudo ./edupi
2. sudo ./edupi -edu
3. sudo ./edupi file
4. sudo ./edupi file -edu
```

Per quanto riguarda il tipo lista il team ha deciso di progettarla eterogenea, non abbiamo dato limiti a quest'ultima permettendo di contenere al suo interno altre liste, funzioni, periferiche. Per facilitare l'utilizzo delle liste ai principianti permettiamo l'inserimento da inizializzazione soltanto di tipi primitivi (*integer,real,string*) mentre attraverso metodi appositi *insert append etc..* è possibile inserire all'interno di una lista anche tipi composti.
Esempio liste: Utilizzando il codice sottostante è possibile definire il metodo range tipico di python, definendo successivamente la funzione printElementList è possibile stampare gli elementi della lista.

```
def range(x){
    list r=[];
    for(integer i=0; i<x; i++){
        r.append(i);
    }
    return r;
}

def printElementList(r){
    for i in r{
    println(strmrg("Element list [", toString(i), "] * 2 =", toString(i*2)));
    }
}
```

Un tipico output da questo codice può essere il seguente:

```
printElementList(range(1));

Output:
   Element list [0] * 2 = 0;
```

Le stringhe sono state definite come array di caratteri quindi possono essere assunte come tipi primitivi, abbiamo fatto questa scelta perchè possiamo accedere facilmente ad ogni elemento della stringa rispetto ad una lista, in cui è facile accedere alla testa rispetto alla posizione n-esima. Il team si è soffermato sulla gestione delle operazioni aritmetiche tra tipi primitivi, quindi per scelta progettuale il team ha deciso di non permettere l'utilizzo delle operazioni per i tipi composti.

4 Caratteristiche del linguaggio

4.1 Grammatica

4.1.1 Tipi

Il linguaggio EduPi possiede sia tipi primitivi che composti.
I tipi primitivi sono:

- `integer`, utilizzato per rappresentare i numeri interi
- `real`, utilizzato per rappresentare i numeri in virgola mobile
- `string`, utilizzato per rappresentare le stringhe di caratteri

I tipi composti sono:

- `peripheral`, utilizzato per rappresentare il tipo periferica
- `list`, utilizzato per rappresentare liste di tipi primitivi e composti

Esempi di dichiarazione dei tipi

```
>integer i;
>real i;
>string i;
>list i;
>peripheral i;
```

Esempi di assegnazione dei tipi

```
>integer i=2;
>real i=2.2;
>string i="EduPi";
>list i=["EduPi",x, 2020];
>peripheral i=["Descrizione", func(x,y)];
```

4.1.2 Espressioni letterali

In questa sezione verranno elencate le operazioni in maniera dettagliata.
Nel linguaggio EduPi è possibile effettuare operazioni su tutti i tipi primitivi, a seconda del tipo verranno effettuate verranno effettuate differenti modalità della stessa operazione.

V1\V2	int	real	string
int	+ - * / or and CMP mod	+ - * / or and CMP	+ - * / or and CMP
real	+ - * / or and CMP	+ - * / or and CMP	+ - * / or and CMP
string	+ - * / or and CMP	+ - * / or and CMP	+ - * / or and CMP

Operazioni consentite dal linguaggio, sono da intendersi come V1 <operatore> V2. Con CMP si indicano le operazioni di comparazione, con mod l'operazione modulare

Operazioni Aritmetiche

Operation + - : Integer +/- Integer = Integer
Integer +/- Real = Real
Integer +/- String = Integer. In questo caso verrà sommato/sottratto un valore intero con il valore rappresentato dalla stringa.
Real +/- Integer = Real
Real +/- Real = Real
Real +/- String = Real. In questo caso verrà sommato/sottratto un valore reale con il valore reale rappresentato dalla stringa.
String +/- String = Real
String +/- Integer = Real
String +/- Real = Real.
Per quanto riguarda queste operazioni con le stringhe valgono le considerazioni fatte precedentemente.

Operation * / : Integer */ Integer = Integer
Integer */ Real = Real
Integer */ String = Real. In questo caso verrà moltiplicato/diviso un valore intero con il valore rappresentato dalla stringa.
Real */ Integer = Real
Real */ Real = Real
Real */ String = Real. In questo caso verrà moltiplicato/diviso un valore intero con il valore rappresentato dalla stringa.
String */ String = Real
String */ Integer = Real
String */ Real = Real.
Per quanto riguarda queste operazioni con le stringhe valgono le considerazioni fatte precedentemente.

Operation mod: Integer % Integer = Integer
E' possibile richiamare l'operazione modulare solo tra tipi Integer.

Operazioni di comparazione
Le operazioni di comparazione in EduPi daranno sempre come risultato un valore intero 1/0. EduPi comprende un serie di operazioni di comparazione:

```
> < != == <= >=
```

Per quanto riguarda le stringhe in base all'operazione di comparazione scelta eseguirà diverse valutazioni.

```
String > String : Valuterà la lunghezza delle stringhe
String < String : Valuterà la lunghezza delle stringhe
String == String : Valuterà il contenuto delle stringhe
String != String : Valuterà il contenuto delle stringhe
String >= String : Valuterà la lunghezza delle stringhe
String <= String : Valuterà la lunghezza delle stringhe
```

Operazioni logiche
Le operazioni logiche in EduPi daranno sempre come risultato un valore intero 1/0. I valori che verranno confrontati verranno valutati come 0 se hanno valore 0 altrimenti verranno valutati come 1 con valori uguali o maggiori di 1. EduPi comprende un serie di operazioni logiche:

```
and or not
```

Operatori unari
EduPi comprende anche alcuni operatori unari:

```
abs: valore assoluto
negate: -valore
{Gli operatori unari non possono essere utilizzati con il tipo string}
```

4.1.3 Strutture di controllo

EduPi comprende comandi condizionali e iterativi.

Comandi condizionali

I comandi condizionali sono previsti da tutti i linguaggi imperativi ed hanno tipicamente la forma:

```
if E then C1 else C2
```

Il team in accordo alle valutazioni fatte precedentemente ha deciso di non stravolgere la struttura tipica di questi comandi.

```
if (cond) { C1 } else { C2 }
if (cond) { C1 }
```

EduPi da la possibilità di innestare più comandi condizionali tra di loro.

Comandi iterativi

Conosciuti con il termine loop. Si classificano in cicli indefiniti e cicli definiti.

Cicli indefiniti
Danno luogo ad un numero di cicli non determinabile a priori. Il team ha deciso di includere:

```
while (cond) { C1 }
do { C1 } while (cond) { C2 }
```

Cicli definiti
Prevedono una variabile di controllo, il team ha deciso di includere:

```
for (init; cond; incr) { C1 }
for var in list { C1 }
```

4.1.4 Funzioni

EduPi da la possibilità al programmatore di creare delle funzioni proprie, sia senza parametri che con parametri, inoltre da anche la possibilità di definire funzioni void o con valori di ritorno.

```
def name(x,y){
    C1
}

def name(x,y){
    C1
    return var
}
```

EduPi presenta inoltre una serie di funzioni built-in che verranno presentate nella sezione successiva.

4.1.5 Selettori

Essendo il tipo periferica, uno dei tipi più importanti nel nostro linguaggio, abbiamo previsto un selettore specifico per accedere agli elementi. Il selettore scelto dal team è "->", con questo selettore è possibile accedere agli elementi contenuti all'interno del tipo periferica.

4.2 Parser

Uno degli strumenti fondamentali per la realizzazione del progetto è sicuramente Bison [3]. Bison è un generatore di parser multiuso che converte una specifica grammatica libera da contesto in un parser LALR(1) o GLR per quella grammatica.In informatica, un parser LALR o un parser LR Look-Ahead è una versione semplificata di un parser LR canonico, per analizzare un testo secondo un insieme di regole di produzione specificate da una grammatica formale ("LR" significa derivazione da sinistra a destra). Grazie a questo strumento è possibile dedicarsi solo ed esclusivamente alla grammatica che si vuole descrivere, concepirla e definirla senza preoccuparsi dei dettagli implementativi. A seguire riportiamo le regole di produzione in formato BNF.

```
<program>::=
| <program> <statement>
| <program> DEF NAME LPAREN <symlist> RPAREN LBRACE <tail> RETURN <exp>
| <program> DEF NAME LPAREN <symlist> RPAREN LBRACE <tail> RBRACE
;

<statement>::= <if_statement>
| <for_statement>
| <for_each>
| <while_statement>
| <do_while_statement>
| <declaration> SEMI
| <init> SEMI
| <exp> SEMI
| <functionV> SEMI
| <pericall> SEMI
;

<if_statement>::= IF LPAREN <exp> RPAREN LBRACE <tail> RBRACE
| IF LPAREN <exp> RPAREN LBRACE <tail> RBRACE ELSE LBRACE <tail> RBRACE
;

<for_statement>::= FOR LPAREN <init> SEMI <exp> SEMI <init> RPAREN LBRACE <tail> RBRACE
;

<for_each>::= FOR NAME IN NAME LBRACE <tail> RBRACE
;

<while_statement>::= WHILE LPAREN <exp> RPAREN LBRACE <tail> RBRACE
;

<do_while_statement>::= DO LBRACE <tail> RBRACE WHILE LPAREN <exp> RPAREN SEMI
;

<tail>::=
| <statement> <tail>
;

<exp>::= <exp> CMP <exp>
| <exp> ADDOP <exp>
| <exp> SUBOP <exp>
| <exp> MULOP <exp>
| <exp> DIVOP <exp>
| <exp> MODOP <exp>
```

```
| <exp> OROP <exp>
| <exp> ANDOP <exp>
| NOTOP <exp>
| ABSOP <exp> ABSOP
| LPAREN <exp> RPAREN
| SUBOP <exp> %prec UMINUS
| <value>
| <functionR>
| NAME LPAREN <explist> RPAREN
;

<declaration>::= <type> NAME
| LST NAME
| PERI NAME
;

<type>::= INT
| STR
| RL
;

<init>::= <type> NAME ASSIGN <exp>
| NAME ASSIGN <exp>
| NAME INCR
| NAME DECR
| LST NAME ASSIGN LBRACK <value_list> RBRACK
| LST NAME ASSIGN <exp>
| LST NAME ASSIGN LBRACK  RBRACK
| NAME ASSIGN LBRACK <value_list> RBRACK
| NAME ASSIGN LBRACK RBRACK
| PERI NAME ASSIGN STRING COMMA <functionlist>
;

<pericall>::= NAME ARR ID
| NAME ARR NAME LPAREN <explist> RPAREN
;

<functionlist>::= <functionlist> COMMA NAME LPAREN <explist> RPAREN
| NAME LPAREN <explist> RPAREN
;

<value>::= NAME
| INTEGER
| REAL
| STRING
;

<functionV>::= PRINT LPAREN <printlist> RPAREN
| PRINTLN LPAREN <printlist> RPAREN
| NAME DOT APP LPAREN <exp> RPAREN
| NAME DOT INS LPAREN <exp> COMMA exp RPAREN
| NAME DOT PUSH LPAREN <exp> RPAREN
| SLP LPAREN <exp> RPAREN
| SOP LPAREN <exp> COMMA <exp> RPAREN
```

```
| RGB LPAREN <exp> COMMA <exp> RPAREN
| INIT LPAREN RPAREN
| SINT LPAREN <exp> RPAREN
| SREAL LPAREN <exp> RPAREN
| SSTR LPAREN <exp> RPAREN
| SLINE LPAREN <exp> RPAREN
| CLEAR LPAREN RPAREN
| STATUS LPAREN RPAREN
| EXIT
;

<functionR>::= TIME LPAREN RPAREN
| NAME DOT POP LPAREN RPAREN
| NAME DOT DEL LPAREN <exp> RPAREN
| NAME DOT GET LPAREN <exp> RPAREN
| NAME DOT SIZE LPAREN RPAREN
| NAME DOT SEARCH LPAREN <exp> RPAREN
| TYPE LPAREN <exp> RPAREN
| SQRT LPAREN <exp> RPAREN
| POW LPAREN <exp> COMMA exp RPAREN
| BUTT LPAREN <exp> RPAREN
| SCAN LPAREN <exp> RPAREN
| STRMRG LPAREN <value_list> RPAREN
| STRMUL LPAREN <exp> COMMA <exp> RPAREN
| TOSTRING LPAREN <exp> RPAREN
| RAND LPAREN <exp> COMMA <exp> RPAREN
| LPAREN INT RPAREN LPAREN <exp> RPAREN
;

<explist>::=
| <exp> COMMA <explist>
| <exp>
;

<printlist>::=
| <exp> CNC <printlist>
| <exp>
;

<value_list>::= <exp> COMMA <value_list>
| <exp>
;

<symlist>::=
| NAME
| NAME COMMA <symlist>
;
```

Il parser presentato non riporta errori o warnings di alcun tipo. Tutte le problematiche riguardo l'ambiguità delle operazioni sono state risolte grazie all'utilizzo degli operatori di precedenza %left %right.

4.3 Lexer

Flex [3](generatore di analizzatori lessicali veloci) è un'alternativa software gratuita e open source a lex. È un programma per computer che genera analizzatori lessicali (noti anche come "scanner" o "lexer"). Per utilizzarlo vengono definite una serie di espressioni regolari che permettono di riconoscere pattern prestabiliti, come costanti, nomi di funzioni, nomi di comandi etc.. La stesse espressioni regolari sono state utilizzate per definire un estensione su Visual Studio Code che permette funzionalità di supporto aggiuntive come il sintax highlighting. E' possibile scaricare l'estensione cercandola nello store con il nome edupi oppure attraverso il seguente link: https://marketplace.visualstudio.com/items?itemName=federicoguida.edupi

Tipi:

```
"list"
"peripheral"
"integer"
"real"
"string"
```

Valori:

```
alpha     [a-zA-Z]
digit     [0-9]
alnum     {alpha}|{digit}
print     [^\"]

NAME      {alpha}+{alnum}*
INTEGER   "0"|[1-9]{digit}*
REAL      "0"|{digit}*"."{digit}+
STRING    \"{print}*\"
```

Strutture di controllo:

```
"if"
"else"
"do"
"while"
"for"
"return"
"def"
"in"
"id"
```

Operatori aritmetici:

```
"+"
"-"
"*"
"/"
"%"
"++"
"--"
```

Operatori di confronto:

```
">"
"<"
"!="
"=="
">="
"<="
```

Operatori logici:

```
"OR"|"or"
"AND"|"and"
"NOT"|"not"
```

Simboli:

```
"|"
"("
")"
"]"
"["
"{"
"}"
";"
"."
","
"="
">>"
```

Selettori:

```
"->"
```

Callbuilt-In:

```
"print"
"println"
"scan"
"time"
"pop"
"push"
"append"
"delete"
"insert"
"get"
"size"
"search"
"sleep"
"type"
"sqrt"
"pow"
"setOutPin"
"ledRGB"
"button"
"lcdInit"
"sendIntegerLcd"
"sendRealLcd"
"sendStringLcd"
"lcdLine"
"clearLcd"
"strmrg"
"strmul"
"toString"
"statusGPIO"
"random"
"exit"
```

Commenti:

```
"//".*  Prende i commenti
[ \t]   Ignora gli spazi bianchi
\\\n    Ignora le linea continue
"\n"    EOL
```

4.4 Funzionalità del linguaggio

EduPi presenta una serie di funzionalità builtin che permettono la manipolazione dei dati, e l'utilizzo delle periferiche, comprendendo ad esempio funzioni time() e sleep() molto utili per la gestione degli input di dati da inviare alla GPIO, permettendo così di mantenere le tempistiche dei protocolli tipici del Raspberry Pi. In questa sezione verranno riportate le funzioni e una descrizione del loro comportamento.

4.4.1 Funzionalità Matematiche

- sqrt(value): Restituisce un real che rappresenta la radice quadrata del valore passato in input
- pow(value): Restituisce un real che rappresenta la potenza del valore passato in input
- random(value): Restituisce un integer compreso tra 0 e il valore passato come input.

4.4.2 Funzionalità Temporali

- time(): Funzione che restituisce la stringa che rappresenta il tempo nel formato year, mon, day, hour, min, sec.
- sleep(value): Questa funzione prende in input sia valori real che integer permettendo così di definire le pause corrette per i protocolli del Raspberry Pi.

4.4.3 Funzionalità Utility

- print(value >> ...): Funzione che prende in input o un singolo valore oppure una serie di valori separati da un delimitatore definito nel linguaggio. Permette quindi di stampare stringhe e valori, liste in maniera veloce e semplice.
- println(value >> ...): Funzione che prende in input o un singolo valore oppure una serie di valori separati da un delimitatore definito nel linguaggio. Permette quindi di stampare stringhe e valori, liste in maniera veloce e semplice. Si differenzia dalla precedente in quanto aggiunge il ritorno a capo.
- scan(tipo): Funzione che prende in input il tipo, in base a quest'ultimo selezionerà la modalità di scan adatta alla memorizzazione del valore.
- exit : Funzione che permette la terminazione del programma.
- (integer)(value): Funzione che permette di fare il cast di un valore appartenente al tipo real a integer.

4.4.4 Funzionalità Stringhe

- strmrg(String1,String2,...,StringN): Prende in input una serie di stringhe per dare come risultato una stringa che rappresenta la concatenazione delle stringhe in input.
- strmul(String,Integer/Real): Funzione che prende in input una stringa e un numero intero o un numero reale opportunamente castato ad integer dalla funzione, per dare come risultato una stringa ripetuta per un numero n di volte definito dal parametro in input.
- toString(Integer/Real): Funzione che prende in input un intero o un numero reale per dare come risultato la stringa che lo rappresenta.

4.4.5 Funzionalità Liste

- list.pop(): Funzione che ritorna la testa della lista, eliminandola.
- list.push(value): Funzione che prende in input in valore e lo mette in testa alla lista.

  Considerazione: Il team ha introdotto queste due funzionalità per permettere ad un programmatore di poter implementare uno stack all'interno del programma.
- list.append(value): Funzione che permette di inserire nella coda della lista qualsiasi tipo di valore(integer, real, string, list, func..);
- list.delete(index): Funzione che elimina e restituisce l'elemento nella posizione indicata dall'indice in input.
- list.insert(index,valore): Funzione che inserisce un determinato elemento all'interno della lista nella posizione indicata dall'indice.
- list.get(index): Funzione che restituisce l'elemento contenuto nell'indice dato come input.
- list.size(): Funzione che ritorna la dimensione della lista.
- list.search(valore): Funzione che ritorna l'indice del primo elemento che corrisponde a quello indicato come input alla funzione.

4.4.6 Funzionalità Periferiche

- setOutPin(pin,level): Funzione che imposta il pin(valore intero) passato come input al livello logico LOW o HIGH in base alla stringa che lo rappresenta.
- ledRGB(pin,intensità[0-100]): Funzione che prende in input un canale RGB e ne modifica l'intensità attraverso il secondo parametro.
- button(pin): Funzione che prende in input il pin che corrisponde alla periferica bottone, restituendo HIGH o LOW in base alla pressione del bottone.
- lcdInit(): Funzione che inizializza la periferica LCD1602 in base al protocollo I2C.
- sendIntegerLcd(Integer): Funzione che invia un valore intero al dispositivo LCD.
- sendRealLcd(Real): Funzione che invia un valore reale al dispositivo LCD.
- sendStringLcd(String): Funzione che invia una stringa al dispositivo LCD.
- lcdLine(String): "LINE1" o "LINE2" permette attraverso l'inserimento di una di queste due stringhe la selezione della riga corrispondente al LCD1602.
- clearLcd(): Funzione che elimina tutti gli elementi rappresentati nello schermo LCD.
- statusGPIO(): Permette di stampare a schermo lo stato della GPIO.

5 Casi d'uso

In questa sezione verranno discussi alcuni casi d'uso per illustrare alcune potenzialità del linguaggio EduPi. EduPi è un linguaggio di programmazione abbastanza completo quindi permette anche la scrittura di programmi molto complessi.

5.1 Implementazione Stack

```
def rot(l) {
    l.append(l.delete(0));
    return l;
}

def swap(l) {
    l.push(l.delete(1));
    return l;
}

def over(l) {
    l.push(l.get(1));
    return l;
}

def nip(l) {
    l.delete(1);
    return l;
}

def dup(l) {
    l.push(l.get(0));
    return l;
}

list stack = [1, 2, 3];
```

Listing 1: Stack

In EduPi è possibile implementare uno stack con tutte le funzioni a disposizione della lista. Attraverso questo programma è possibile implementare i metodi più utilizzati per la manipolazione di uno stack.

5.2 Utilizzo periferiche

```
// functions //

def ledOn(pin) {
    setOutPin(pin, "HIGH");
}

def ledOff(pin) {
    setOutPin(pin, "LOW");
}

def buzzOn(pin) {
    setOutPin(pin, "HIGH");
}

def buzzOff(pin) {
    setOutPin(pin, "LOW");
}

```

```
19 def ledBlink(pin, n) {
20     integer x = pin;
21     for(integer i = 1; i <= n; i++) {
22         ledOn(x);
23         sleep(500);
24         ledOff(x);
25         sleep(500);
26     }
27 }
28
29 def buzzBlink(pin, n) {
30     integer x = pin;
31     for(integer i = 1; i <= n; i++) {
32         buzzOn(x);
33         sleep(500);
34         buzzOff(x);
35         sleep(500);
36     }
37 }
38
39 // INIT //
40
41 peripheral led = ["LED", ledOn(), ledOff(), ledBlink()];
42 peripheral buzzer = ["BUZZER", buzzOn(),
43                 buzzOff(), buzzBlink()];
```

Listing 2: Dichiarazione periferiche

Attraverso questo esempio di codice il team vuole mostrare una delle potenzialità di EduPi. Il professore può preparare il programma definendo tutti i comportamenti delle periferiche. Successivamente può distribuire questo codice a tutti gli studenti. Durante la lezione così grazie all'utilizzo di EduPi gli studenti possono utilizzare tutti i metodi preparati dal professore in modalità interattiva. Il codice verrà lanciato in modalità sicura permettendo una supervisione sulle periferiche.

5.3 Comportamenti complessi periferiche

```
1  integer count = 0;
2
3  def beep(note, duration) {
4      real beepDelay = (1000/note);
5      real tempo = (duration*1000)/(beepDelay*2*1000);
6
7      for(integer i = 0; i < tempo; i++) {
8          setOutPin(24, "HIGH");
9          sleep(beepDelay);
10         setOutPin(24, "LOW");
11         sleep(beepDelay);
12     }
13     setOutPin(24, "LOW");
14     sleep(20);
15     if(count % 2 == 0) {
16         ledRGB(16, 0);
17         ledRGB(20, 0);
18         ledRGB(21, 100);
```

```
19      }else {
20          ledRGB(16, 100);
21          ledRGB(20, 0);
22          ledRGB(21, 100);
23      }
24      count++;
25  }
26
27    beep(440, 500);
28    beep(440, 500);
29    beep(349, 350);
30    beep(523, 150);
31
32    beep( 440, 500);
33    beep( 349, 350);
34    beep( 523, 150);
35    beep( 440, 1000);
36    beep( 659, 500);
37
38    beep( 659, 500);
39    beep( 659, 500);
40    beep( 698, 350);
41    beep( 523, 150);
42    beep( 415, 500);
43
44    beep( 349, 350);
45    beep( 523, 150);
46    beep( 440, 1000);
47    beep( 880, 500);
48    beep( 440, 350);
49
50    beep( 440, 150);
51    beep( 880, 500);
52    beep( 830, 250);
53    beep( 784, 250);
54    beep( 740, 125);
55
56    beep( 698, 125);
57    beep( 740, 250);
58
59    sleep(250);
60
61    beep( 455, 250);
62    beep( 622, 500);
63    beep( 587, 250);
64    beep( 554, 250);
65    beep( 523, 125);
66
67    beep( 466, 125);
68    beep( 523, 250);
69
70    sleep(250);
71
72    beep( 349, 125);
```

```
73    beep( 415, 500);
74    beep( 349, 375);
75    beep( 440, 125);
76    beep( 523, 500);
77
78    beep( 440, 375);
79    beep( 523, 125);
80    beep( 659, 1000);
81    beep( 880, 500);
82    beep( 440, 350);
83
84    beep( 440, 150);
85    beep( 880, 500);
86    beep( 830, 250);
87    beep( 784, 250);
88    beep( 740, 125);
89
90    beep( 698, 125);
91    beep( 740, 250);
92
93    sleep(250);
94
95    beep( 455, 250);
96    beep( 622, 500);
97    beep( 587, 250);
98    beep( 554, 250);
99    beep( 523, 125);
100
101   beep( 466, 125);
102   beep( 523, 250);
103
104   sleep(250);
105
106   beep( 349, 250);
107   beep( 415, 500);
108   beep( 349, 375);
109   beep( 523, 125);
110   beep( 440, 500);
111
112   beep( 349, 375);
113   beep( 261, 125);
114   beep( 440, 1000);
115
116   ledRGB(16, 100);
117   sleep(50);
118   ledRGB(20, 100);
119   sleep(50);
120   ledRGB(21, 100);
```

Listing 3: Utilizzo complesso delle periferiche

EduPi permette grazie alla presenza di numerose funzioni e operatori di definire comportamenti complessi alle periferiche. Attraverso queste righe di codice è possibile creare una canzone basandosi sulla frequenza dei suoni, affiancato con il led rgb che ad ogni cambio di nota presenta un colore differente.

5.4 Modalità sicura

Questo caso d'uso mostra le potenzialità della modalità sicura.
Supponiamo che il codice del Listing 2 sia stato eseguito attraverso il seguente comando:

```
sudo ./edupi periOut.edupi -edu
```

Da questo momento lo studente avrà a disposizione le periferiche definite insieme ai comportamenti. La prima cosa che potrebbe fare è stampare le informazioni sulla periferica.

```
println(led);
println(buzzer);
```

Output:

```
Peripheral Description: "LED"
Method --> ledOn(pin)
Method --> ledOff(pin)
Method --> ledBlink(pin,n)

Peripheral Description: "BUZZER"
Method --> buzzOn(pin)
Method --> buzzOff(pin)
Method --> buzzBlink(pin,n)
```

Lo studente ora ha tutte le informazioni riguardo i metodi disponibili per ciascuna periferica.
Esempio[1]:

```
led->buzzOn(12);
```

Output:

```
Method not found!
```

Se lo studente richiama un metodo non presente all'interno della periferica verrà lanciato un messaggio di avvertimento.

Esempio[2]:

```
buzzer->buzzOn(2);
```

Output:

```
I2C PIN SELECTED
```

In modalità -edu il linguaggio EduPi rispetterà le convenzioni della BCM, garantendo così che i pin richiamati siano adeguati alla funzione che si vuole utilizzare.

Esempio[3]:

```
buzzer->buzzOn(50);
```

Output:

```
Undefined Pin, ground or power supply pin
```

Questo messaggio avvisa lo studente che il pin selezionato non è presente all'interno della GPIO, oppure è un pin di ground o di power supply.

6 Risultati

Per analizzare il linguaggio EduPi in termini prestazionali sono stati utilizzati alcuni algoritmi famosi per il calcolo computazione di un linguaggio di programmazione.

6.1 PiGreco

Abbiamo utilizzato l'algoritmo del calcolo del pigreco

```
1 real accum = 0;
2 integer n = 1000000;
3
4 for(integer i = 0; i < n; i++) {
5     accum = accum + (pow(-1, i)/(2*i+1));
6 }
7
8 print(accum*4);
```

Listing 4: Calcolo pigrego EduPi

```
1 #!/usr/bin/python3
2
3 accum = 0;
4 n = 100000;
5
6 for i in range(n):
7     accum += (pow(-1, i)/(2i+1));
8
9 print(accum4);
```

Listing 5: Calcolo pigrego Python

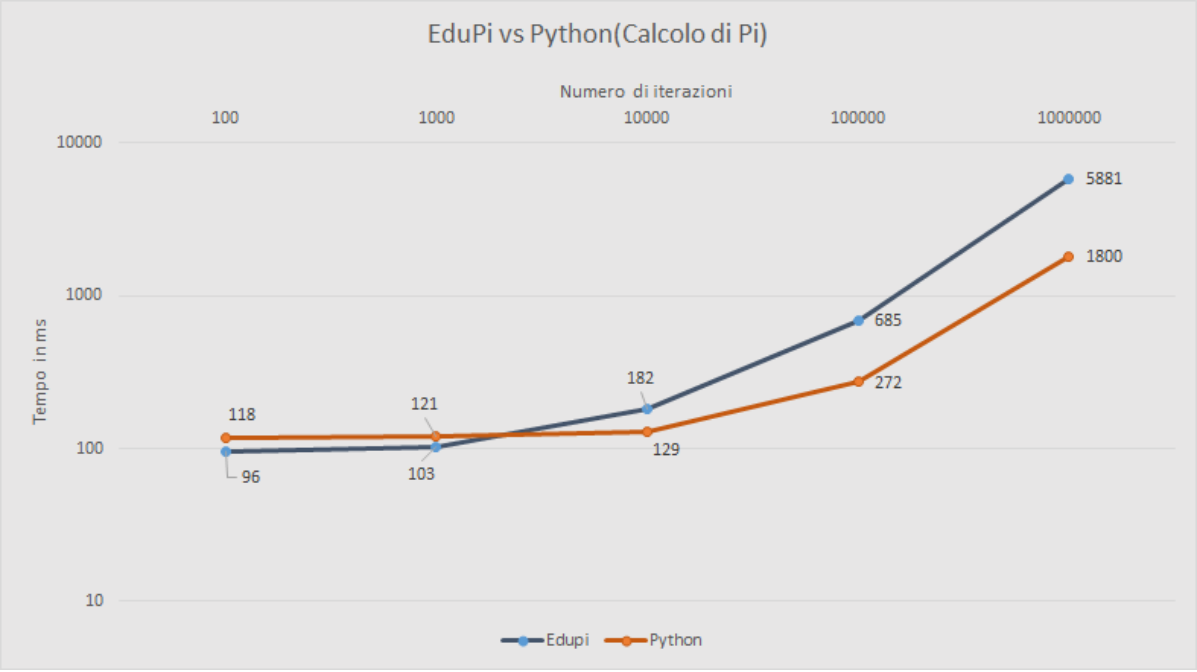

Figure 1: Grafico confronto EduPi vs Python

In questa analisi abbiamo confrontato il linguaggio EduPi con Python.Come si evince dal grafico il linguaggio EduPi ha una crescita esponenziale maggiore rispetto a Python, come ci aspettavamo. Fino a 1000 iterazioni

il linguaggio EduPi riesce a tenere testa a Python, il team nelle prossime versioni di EduPi cercherà di ottimizzarlo cercando di aumentare le prestazioni al crescere del numero di iterazioni.

6.2 Confronto strutture EduPi

Oltre a confrontare il linguaggio EduPi con altri linguaggi famosi dell'ultimo decennio, il team ha analizzato e confrontato EduPi con se stesso in base alle strutture utilizzate. Il team per effettuare questa analisi ha deciso di utilizzare un determinato algoritmo implementato con diverse strutture di controllo permettendo di raccogliere statistiche temporali grazie all'utilizzo del comando bash *time* preposto all'esecuzione di EduPi.

```
1  def range(x) {
2      list l = [];
3      integer i = 0;
4      while(x > l.size()) {
5          l.append(i);
6          i++;
7      }
8      return l;
9  }
10
11 list t = range(10000);
12 list z = range(10000-2);
13
14 for i in z {
15     t.pop();
16 }
17
18 println(t);
```

Listing 6: Codice For Each

```
1  def range(x) {
2      list l = [];
3      integer i = 0;
4      while(x > l.size()) {
5          l.append(i);
6          i++;
7      }
8      return l;
9  }
10
11 integer y = 10000;
12 list t = range(y);
13
14 for(integer n = 0; n < (y-2); n++) {
15     t.pop();
16 }
17 println(t);
```

Listing 7: Codice For

```
1  def range(x) {
2      list l = [];
3      integer i = 0;
4      while(x > l.size()) {
```

```
5           l.append(i);
6           i++;
7       }
8       return l;
9   }
10
11  list t = range(10000);
12
13  while(t.size() > 2) {
14      t.pop();
15  }
```

Listing 8: Codice While

```
1   def range(x) {
2       list l = [];
3       integer i = 0;
4       while(x > l.size()) {
5           l.append(i);
6           i++;
7       }
8       return l;
9   }
10
11  def empty(y) {
12      while(y.size() > 2) {
13          y.pop();
14      }
15      return y;
16  }
17
18  list t = range(10000);
19  println(empty(t));
```

Listing 9: Codice Funzione

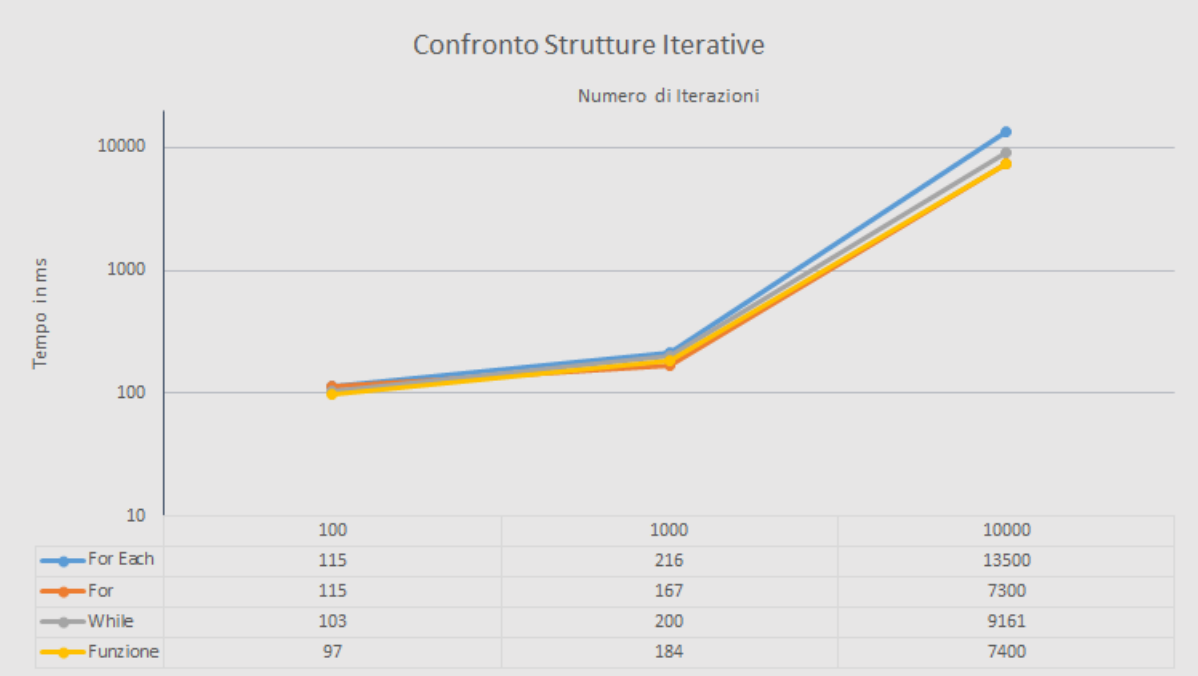

Figure 2: Grafico confronto EduPi vs EduPi

Dal risultato dell'analisi si evince che l'algoritmo implementato con l'utilizzo di una funzione risulta essere il più veloce all'aumentare delle iterazioni. Mentre il For Each risulta essere la struttura meno performante.

7 Conclusioni

In conclusione EduPi risulta essere un linguaggio abbastanza completo, oltre ad essere utilizzabile per il suo reale scopo riesce a spaziare su svariati campi. EduPi permette la scrittura di programmi abbastanza complessi e l'utilizzo di periferiche complesse con poche righe di codice, soffermando così l'attenzione sull'insegnamento scolastico. Il team nelle future versioni di EduPi introdurrà alcuni aspetti come lo scope delle variabili e l'ottimizzazione delle performance e della memoria, cercando di irrobustire l'intero linguaggio e l'infrastruttura attorno. Inoltre nelle successive versioni verranno introdotte altre periferiche e protocolli tra i più utilizzati per il Raspberry Pi come SPI e UART.

References

[1] Narasimha Saii Yamanoor and Srihari Yamanoor. High quality, low cost education with the raspberry pi. In *2017 IEEE Global Humanitarian Technology Conference (GHTC)*, pages 1–5. IEEE, 2017.

[2] Nicholas Nethercote and Julian Seward. Valgrind: A program supervision framework. *Electronic notes in theoretical computer science*, 89(2):44–66, 2003.

[3] John Levine. *Flex & Bison: Text Processing Tools*. " O'Reilly Media, Inc.", 2009.