ASCENSION

Linguaggio di Programmazione

Manuale Utente Completo

Versione 12.6 (Substr Edition)

Autore: EdeFede

NOTA IMPORTANTE

*Ascension è un linguaggio di programmazione sviluppato per **uso didattico e hobbistico**. È pensato per chi vuole imparare i concetti della programmazione in modo semplice e diretto, sperimentando con una sintassi pulita e intuitiva.*

*Ascension **È la sintassi**: l'implementazione attuale in Python è solo una delle possibili realizzazioni. Il vero valore sta nel design del linguaggio stesso, che può essere implementato in qualsiasi altro linguaggio (C, C++, Rust, ecc.) mantenendo la stessa semantica e comportamento.*

Indice

1. Introduzione

Ascension è un linguaggio di programmazione moderno con sintassi C-style, progettato per essere semplice da imparare ma sufficientemente potente per creare applicazioni reali. È implementato come una macchina virtuale stack-based con un compilatore che traduce il codice sorgente in bytecode.

Caratteristiche Principali

- Sintassi pulita ispirata al C, senza dichiarazioni di tipo esplicite
- Macchina virtuale stack-based efficiente
- Supporto per strutture dati (struct) personalizzate
- Array monodimensionali e matrici bidimensionali
- Funzioni con scope locale e supporto per ricorsione
- Gestione eccezioni con try/catch
- I/O su file completo
- Networking HTTP (GET/POST) e Socket TCP
- Interfaccia grafica tramite Tkinter
- Interfaccia terminale tramite Curses
- Esecuzione comandi di sistema
- Sistema di include per codice modulare

2. Installazione e Utilizzo

Requisiti

Per eseguire programmi Ascension è necessario Python 3.6 o superiore. Alcune funzionalità opzionali richiedono librerie aggiuntive:

- **requests** - per funzionalità HTTP (http_get, http_post)
- **tkinter** - per interfaccia grafica (solitamente incluso in Python)

Esecuzione

Per eseguire un programma Ascension:

```
python ascension_12_6.py programma.asc
```

Per attivare la modalità debug (visualizza il bytecode generato):

```
python ascension_12_6.py programma.asc -debug
```

Primo Programma

Crea un file chiamato **hello.asc**:

```
// Il classico Hello World
print("Ciao, Mondo!");
```

3. Sintassi di Base

Commenti

Ascension supporta commenti su singola riga e multi-riga:

```
// Questo è un commento su singola riga

/* Questo è un commento
   su più righe */
```

Variabili

Le variabili in Ascension non richiedono dichiarazione di tipo. Vengono create automaticamente al primo assegnamento:

```
nome = "Mario";
eta = 25;
altezza = 1.75;
attivo = true;
```

Variabili Globali

Per dichiarare una variabile globale all'interno di una funzione:

```
func incrementa() {
    global contatore = contatore + 1;
}
```

Statement

Ogni istruzione termina con un punto e virgola. I blocchi di codice sono delimitati da parentesi graffe:

```
x = 10;
if (x &gt; 5) {
    print("x è maggiore di 5");
}
```

4. Tipi di Dati

Tipi Primitivi

Tipo	Esempio	Descrizione
Intero	`42, -17, 0`	Numeri interi
Float	`3.14, -0.5, 1.0`	Numeri decimali
Stringa	`"Hello", "Ciao"`	Testo tra virgolette doppie
Booleano	`true, false`	Valori logici (1 e 0)
NULL	`NULL`	Valore speciale per assenza/errore

Stringhe

Le stringhe supportano sequenze di escape:

```
testo = "Prima riga\nSeconda riga";
tab = "Colonna1\tColonna2";
quote = "Disse: \"Ciao!\"";
```

Sequenze di escape supportate:

- `\n` - nuova riga
- `\t` - tabulazione
- `\r` - ritorno carrello
- `\"` - virgolette doppie
- `\\` - backslash

NULL

NULL è un valore speciale che indica assenza di valore o condizione di errore. È diverso da 0 e dalla stringa vuota:

```
risultato = NULL;
if (risultato == NULL) {
    print("Nessun risultato");
}
```

5. Operatori

Operatori Aritmetici

Operatore	Descrizione	Esempio
`+`	Addizione / Concatenazione stringhe	`a + b, "Hello" + " World"`
`-`	Sottrazione	`a - b`
`*`	Moltiplicazione	`a * b`
`/`	Divisione	`a / b`
`%`	Modulo (resto)	`a % b`

Operatori di Confronto

Operatore	Descrizione	Esempio
`==`	Uguale	`a == b`
`!=`	Diverso	`a != b`
`>`	Maggiore	`a > b`
`<`	Minore	`a < b`
`>=`	Maggiore o uguale	`a >= b`
`<=`	Minore o uguale	`a <= b`

Operatori Logici

Operatore	Descrizione	Esempio
`&&`	AND logico	`a && b`
`||`	OR logico	`a || b`
`!`	NOT logico	`!a`

Operatori Composti di Assegnamento

Ascension supporta gli operatori composti:

```
x += 5;   // equivale a: x = x + 5
x -= 3;   // equivale a: x = x - 3
x *= 2;   // equivale a: x = x * 2
x /= 4;   // equivale a: x = x / 4
```

6. Strutture di Controllo

If / Else If / Else

```
x = 10;

if (x &gt; 10) {
    print("Maggiore di 10");
} else if (x == 10) {
    print("Uguale a 10");
} else {
    print("Minore di 10");
}
```

While

```
i = 0;
while (i &lt; 5) {
    print(i);
    i += 1;
}
```

For

Il ciclo for ha la sintassi classica C-style:

```
for (i = 0; i &lt; 10; i += 1) {
    print(i);
}
```

Switch

```
scelta = 2;

switch (scelta) {
    case 1: {
        print("Scelta uno");
    };
    case 2: {
        print("Scelta due");
    };
    case 3: {
        print("Scelta tre");
    };
    default: {
        print("Scelta non valida");
    };
}
```

Break e Continue

```
for (i = 0; i &lt; 10; i += 1) {
    if (i == 3) {
        continue;  // Salta iterazione
    }
    if (i == 7) {
        break;     // Esce dal loop
    }
    print(i);
}
```

7. Funzioni

Definizione di Funzioni

```
func saluta(nome) {
    print("Ciao, " + nome + "!");
}

saluta("Mario");  // Output: Ciao, Mario!
```

Funzioni con Valore di Ritorno

```
func somma(a, b) {
    return a + b;
}

risultato = somma(5, 3);
print(risultato);  // Output: 8
```

Ricorsione

```
func fattoriale(n) {
    if (n &lt;= 1) {
        return 1;
    }
    return n * fattoriale(n - 1);
}

print(fattoriale(5));  // Output: 120
```

Prototipi di Funzione (Forward Declarations)

I prototipi permettono di dichiarare una funzione prima della sua definizione. Questo è utile per la ricorsione mutua o per organizzare meglio il codice:

```
// Dichiarazione del prototipo (senza corpo)
func pari(n);
func dispari(n);

// Definizione delle funzioni
func pari(n) {
    if (n == 0) { return true; }
    return dispari(n - 1);
}

func dispari(n) {
    if (n == 0) { return false; }
    return pari(n - 1);
}

print(pari(4));   // Output: 1 (true)
print(dispari(4)); // Output: 0 (false)
```

8. Strutture Dati (Struct)

Definizione di Struct

```
struct Persona { nome, eta, citta }

// Creazione di un'istanza
mario = new Persona();
mario.nome = "Mario";
mario.eta = 30;
mario.citta = "Roma";

print(mario.nome + " ha " + mario.eta + " anni");
```

Struct Annidati

```
struct Indirizzo { via, numero, cap }
struct Cliente { nome, indirizzo }

cliente = new Cliente();
cliente.nome = "Luigi";
cliente.indirizzo = new Indirizzo();
cliente.indirizzo.via = "Via Roma";
cliente.indirizzo.numero = 42;
cliente.indirizzo.cap = "00100";
```

Dizionari (Dict)

È possibile creare dizionari con sintassi letterale:

```
// Dizionario vuoto
config = {};

// Dizionario con valori
persona = {nome: "Anna", eta: 25, attivo: true};

// Accesso ai valori
print(persona.nome);
persona.eta = 26;

// Aggiunta di nuove chiavi
persona.email = "anna@email.com";
```

9. Array e Matrici

Array Monodimensionali

```
// Creazione di un array
numeri[0] = 10;
numeri[1] = 20;
numeri[2] = 30;

// Accesso agli elementi
print(numeri[1]);  // Output: 20

// Iterazione
for (i = 0; i &lt; 3; i += 1) {
    print(numeri[i]);
}
```

Matrici (Array 2D)

Ascension supporta due sintassi per le matrici:

```
// Sintassi C-style
matrice[0][0] = 1;
matrice[0][1] = 2;
matrice[1][0] = 3;
matrice[1][1] = 4;

// Sintassi con virgola
matrice[0, 0] = 1;
matrice[0, 1] = 2;
matrice[1, 0] = 3;
matrice[1, 1] = 4;

// Accesso
valore = matrice[1][0];  // o matrice[1, 0]
```

Funzione matrix()

Crea una matrice inizializzata:

```
// Crea matrice 3x4 inizializzata a 0
m = matrix(3, 4, 0);

// Crea matrice 2x2 inizializzata a 1
identita = matrix(2, 2, 1);

// Funzioni per dimensioni
righe = rows(m);   // 3
colonne = cols(m); // 4
dimensione = dim(m); // 2 (bidimensionale)
```

Funzioni per Array

```
arr[0] = "primo";
arr[1] = "secondo";
arr[2] = "terzo";

lunghezza = len(arr);  // 3
chiavi = keys(arr);    // Array con [0, 1, 2]
```

10. Gestione File

Apertura File

La funzione **open()** apre un file e restituisce un handle. Le modalità supportate sono:

- `"r"` - lettura
- `"w"` - scrittura (sovrascrive)
- `"a"` - append (aggiunge in fondo)

Scrittura su File

```
// Aprire file in scrittura
f = open("output.txt", "w");

if (f != NULL) {
    write(f, "Prima riga\n");
    write(f, "Seconda riga\n");
    close(f);
    print("File scritto con successo");
} else {
    print("Errore apertura file");
}
```

Lettura da File

```
// Lettura riga per riga
f = open("input.txt", "r");
if (f != NULL) {
    linea = read_line(f);
    while (len(linea) &gt; 0) {
        print(linea);
        linea = read_line(f);
    }
    close(f);
}

// Lettura completa
f = open("input.txt", "r");
if (f != NULL) {
    contenuto = read_all(f);
    print(contenuto);
    close(f);
}
```

Funzioni File I/O

Funzione	Descrizione	Ritorno
`open(file, modo)`	Apre un file	`handle o NULL`
`write(handle, testo)`	Scrive nel file	`1 o NULL`
`read_line(handle)`	Legge una riga	`stringa o NULL`
`read_all(handle)`	Legge tutto il file	`stringa o NULL`
`close(handle)`	Chiude il file	`1 o NULL`

11. Input/Output

Output con print()

```
// Stampa semplice
print("Hello World!");

// Stampa multipla
nome = "Mario";
eta = 30;
print("Nome:", nome, "Età:", eta);

// Concatenazione
print("Il risultato è: " + (5 + 3));
```

Input con read()

```
print("Come ti chiami?");
nome = read();
print("Ciao, " + nome + "!");

print("Quanti anni hai?");
eta = to_int(read());
print("Tra 10 anni ne avrai " + (eta + 10));
```

Conversioni di Tipo

```
// Stringa a intero
numero = to_int("42");

// Stringa a float
decimale = to_float("3.14");

// Carattere a codice ASCII
codice = to_int("A");  // 65

// Codice ASCII a carattere
carattere = chr(65);   // "A"

// Estrazione sottostringa
testo = "Hello World";
sub = substr(testo, 0, 5);  // "Hello"
sub = substr(testo, 6, 5);  // "World"
```

12. Gestione Errori (Try/Catch)

Try/Catch Base

```
try {
    risultato = 10 / 0;  // Errore!
} catch (e) {
    print("Errore catturato: " + e);
}
```

Throw

È possibile lanciare eccezioni personalizzate:

```
func dividi(a, b) {
    if (b == 0) {
        throw "Divisione per zero!";
    }
    return a / b;
}

try {
    risultato = dividi(10, 0);
} catch (errore) {
    print("Errore: " + errore);
}
```

Try/Catch senza Variabile

```
try {
    operazioneRischiosa();
} catch {
    print("Si è verificato un errore");
}
```

13. Networking (HTTP)

Richiede la libreria Python 'requests'

HTTP GET

```
risposta = http_get("https://api.example.com/data");

status = response_status(risposta);
body = response_body(risposta);

if (status == 200) {
    print("Risposta: " + body);
} else {
    print("Errore HTTP: " + status);
}
```

HTTP POST

```
dati = {nome: "Mario", email: "mario@test.it"};

risposta = http_post("https://api.example.com/users", dati);

if (response_status(risposta) == 201) {
    print("Utente creato!");
}
```

Funzioni HTTP

Funzione	Descrizione	Ritorno
`http_get(url)`	Esegue richiesta GET	`oggetto risposta`
`http_post(url, dati)`	Esegue richiesta POST	`oggetto risposta`
`response_status(r)`	Codice stato HTTP	`intero`
`response_body(r)`	Corpo della risposta	`stringa`

14. Socket TCP

Risoluzione DNS

```
ip = get_ip("www.google.com");
print("IP di Google: " + ip);
```

Client TCP

```
// Creazione socket
sock = socket_open("TCP", "IPv4");

if (sock != NULL) {
    // Connessione
    if (socket_connect(sock, "93.184.216.34", 80) != NULL) {
        // Invio richiesta
        socket_send(sock, "GET / HTTP/1.1\r\nHost: example.com\r\n\r\n");

        // Ricezione risposta
        risposta = socket_recv(sock, 4096);
        print(risposta);
    }
    socket_close(sock);
}
```

Server TCP

```
// Creazione socket server
server = socket_open("TCP", "IPv4");

if (server != NULL) {
    socket_bind(server, "0.0.0.0", 8080);
    socket_listen(server, 5);
    print("Server in ascolto sulla porta 8080...");

    // Accetta connessione
    client = socket_accept(server);
    if (client != NULL) {
        dati = socket_recv(client, 1024);
        print("Ricevuto: " + dati);
        socket_send(client, "Messaggio ricevuto!");
        socket_close(client);
    }
    socket_close(server);
}
```

Funzioni Socket

Funzione	Descrizione	Ritorno
`socket_open(tipo, proto)`	Crea socket (TCP/UDP, IPv4)	`handle o NULL`
`socket_bind(s, ip, porta)`	Associa indirizzo al socket	`1 o NULL`
`socket_listen(s, backlog)`	Mette socket in ascolto	`1 o NULL`
`socket_accept(s)`	Accetta connessione	`nuovo handle o NULL`
`socket_connect(s, ip, porta)`	Connette a server	`1 o NULL`

`socket_send(s, dati)`	Invia dati	`byte inviati o NULL`
`socket_recv(s, maxbyte)`	Riceve dati	`stringa o NULL`
`socket_close(s)`	Chiude socket	`1 o NULL`
`get_ip(hostname)`	Risolve hostname in IP	`stringa IP`

15. Interfaccia Grafica (Tkinter)

Richiede tkinter installato in Python

Finestra Base

```
// Creazione finestra principale
root = tk_root("La mia App");
tk_geometry("400x300");

// Creazione etichetta
label = tk_widget(root, "Label", {text: "Benvenuto!"});
tk_pack(label, {pady: 20});

// Creazione bottone
btn = tk_widget(root, "Button", {text: "Clicca qui"});
tk_command(btn, "onClick");
tk_pack(btn, {});

// Funzione callback
func onClick() {
    tk_msgbox("Info", "Hai cliccato il bottone!");
}

// Avvio loop eventi
tk_mainloop();
```

Widget Disponibili

Label	Etichetta di testo	{text: "Testo"}
`Button`	Pulsante cliccabile	`{text: "OK"}`
`Entry`	Campo di input singola riga	`{width: 30}`
`Text`	Area di testo multilinea	`{width: 40, height: 10}`
`Listbox`	Lista di elementi	`{width: 20, height: 5}`
`Canvas`	Area di disegno	`{width: 400, height: 300}`
`Frame`	Contenitore per altri widget	`{}`

Input e Dialog

```
// Campo di input
entry = tk_widget(root, "Entry", {width: 30});
tk_pack(entry, {});

// Leggere valore
valore = tk_get(entry);

// Impostare valore
tk_set(entry, "Nuovo testo");

// Dialog per file
file = tk_filedialog_open("Seleziona file");
file = tk_filedialog_save("Salva come");

// Dialog per input
nome = tk_askstring("Input", "Come ti chiami?");

// Dialog Sì/No
if (tk_askyesno("Conferma", "Sei sicuro?") == 1) {
    print("Confermato!");
}
```

Canvas e Disegno

```
canvas = tk_widget(root, "Canvas", {width: 400, height: 300, bg: "white"});
tk_pack(canvas, {});

// Disegno forme
linea = tk_canvas_line(canvas, 10, 10, 100, 100, "blue");
rett = tk_canvas_rect(canvas, 50, 50, 150, 100, "red");
cerchio = tk_canvas_oval(canvas, 200, 50, 280, 130, "green");
testo = tk_canvas_text(canvas, 200, 200, "Hello", "black");

// Spostamento
tk_canvas_move(canvas, rett, 20, 10);

// Cancellazione
tk_canvas_delete(canvas, linea);
tk_canvas_clear(canvas);  // Cancella tutto
```

Eventi e Bind

```
// Bind evento tastiera
tk_bind(root, "&lt;Key&gt;", "onKeyPress");

func onKeyPress(evento) {
    print("Tasto premuto: " + evento.key);
    print("Codice: " + evento.keycode);
}

// Bind evento mouse
tk_bind(canvas, "&lt;Button-1&gt;", "onClick");

func onClick(evento) {
    print("Click a x=" + evento.x + " y=" + evento.y);
}

// Timer
afterId = tk_after(1000, "onTimer");  // Dopo 1 secondo
tk_after_cancel(afterId);  // Cancella timer
```

Funzioni Tkinter Complete

Funzione	Descrizione	Ritorno
`tk_root(titolo)`	Crea finestra principale	`handle`
`tk_widget(parent, tipo, config)`	Crea widget	`handle`
`tk_pack(widget, config)`	Posiziona con pack	`1`
`tk_grid(widget, config)`	Posiziona con grid	`1`
`tk_config(widget, config)`	Modifica proprietà	`1`
`tk_command(widget, func)`	Associa callback	`1`
`tk_bind(widget, evento, func)`	Associa evento	`1`
`tk_get(widget)`	Ottiene valore Entry	`stringa`
`tk_set(widget, valore)`	Imposta valore Entry/Text	`1`

`tk_clear(widget)`	Pulisce widget	`1`
`tk_focus(widget)`	Imposta focus	`1`
`tk_destroy(widget)`	Distrugge widget	`1`
`tk_geometry(dim)`	Imposta dimensioni finestra	`1`
`tk_title(titolo)`	Cambia titolo finestra	`1`
`tk_resizable(w, h)`	Abilita/disabilita resize	`1`
`tk_msgbox(titolo, msg)`	Mostra messaggio	`1`
`tk_askstring(titolo, prompt)`	Input dialog	`stringa`
`tk_askyesno(titolo, msg)`	Dialog Sì/No	`1 o 0`
`tk_filedialog_open(titolo)`	Dialog apri file	`percorso`
`tk_filedialog_save(titolo)`	Dialog salva file	`percorso`
`tk_text_get(widget)`	Ottiene testo da Text	`stringa`
`tk_text_insert(w, pos, txt)`	Inserisce in Text	`1`
`tk_listbox_add(widget, item)`	Aggiunge a Listbox	`1`
`tk_listbox_get(widget)`	Elemento selezionato	`stringa`
`tk_listbox_index(widget)`	Indice selezionato	`intero`
`tk_after(ms, func)`	Timer	`id`
`tk_after_cancel(id)`	Cancella timer	`1`
`tk_update()`	Aggiorna interfaccia	`1`
`tk_mainloop()`	Avvia loop eventi	`1`

16. Interfaccia Terminale (Curses)

Curses permette di creare interfacce testuali interattive nel terminale. È utile per giochi testuali, menu interattivi, e applicazioni console avanzate.

Esempio Base

```
curses_init();
curses_clear();

curses_move(5, 10);
curses_write("Premi un tasto per uscire...");
curses_refresh();

tasto = curses_read_key();
print("Hai premuto il tasto con codice: " + tasto);

curses_end();
```

Funzioni Curses

Funzione	Descrizione	Ritorno
`curses_init()`	Inizializza modalità curses	`1 o 0`
`curses_end()`	Termina modalità curses	`1 o 0`
`curses_clear()`	Pulisce lo schermo	`1 o 0`
`curses_refresh()`	Aggiorna lo schermo	`1 o 0`
`curses_move(y, x)`	Sposta cursore	`1 o 0`
`curses_write(testo)`	Scrive alla posizione cursore	`1 o 0`
`curses_read_key()`	Legge un tasto	`codice tasto`

17. Comandi di Sistema

Esecuzione Comandi Shell

```
// Esegue comando e ottiene exit code
exitCode = system("ls -la");
if (exitCode == 0) {
    print("Comando eseguito con successo");
}

// Esegue comando e cattura output
output = exec("date");
print("Data corrente: " + output);

// Esempio: lista file
files = exec("ls");
print(files);
```

Funzioni Sistema

Funzione	Descrizione	Ritorno
`system(comando)`	Esegue comando shell	`exit code`
`exec(comando)`	Esegue e cattura output	`stringa output`

Attenzione: *L'esecuzione di comandi di sistema può essere pericolosa. Usare con cautela e mai con input non validato dall'utente.*

18. Include e Modularità

La direttiva **include** permette di includere codice da altri file Ascension. Questo consente di organizzare il codice in moduli riutilizzabili.

Sintassi

```
include "percorso/file.asc";
```

Esempio: Libreria Matematica

File **math_lib.asc**:

```
// math_lib.asc - Funzioni matematiche

func abs(n) {
    if (n &lt; 0) {
        return n * -1;
    }
    return n;
}

func max(a, b) {
    if (a &gt; b) { return a; }
    return b;
}

func min(a, b) {
    if (a &lt; b) { return a; }
    return b;
}

func potenza(base, exp) {
    risultato = 1;
    for (i = 0; i &lt; exp; i += 1) {
        risultato *= base;
    }
    return risultato;
}
```

File principale **main.asc**:

```
include "math_lib.asc";

print(abs(-42));        // Output: 42
print(max(10, 20));     // Output: 20
print(potenza(2, 8));   // Output: 256
```

Organizzazione Consigliata

Per progetti più grandi, si consiglia di organizzare i file in questo modo:

```
progetto/
■■■ main.asc           # Punto di ingresso
■■■ lib/
■   ■■■ utils.asc      # Funzioni utility
■   ■■■ math.asc       # Funzioni matematiche
■   ■■■ io.asc         # Funzioni I/O
■■■ config.asc         # Configurazioni
```

19. Funzioni Built-in Complete

I/O e Conversioni

Funzione	Descrizione	Ritorno
`print(arg1, arg2, ...)`	Stampa valori a schermo	`-`
`read()`	Legge input da tastiera	`stringa`
`to_int(valore)`	Converte a intero	`intero`
`to_float(valore)`	Converte a float	`float`
`chr(codice)`	Codice ASCII a carattere	`stringa`

Stringhe e Array

Funzione	Descrizione	Ritorno
`len(obj)`	Lunghezza stringa/array	`intero`
`keys(array)`	Chiavi di un array/dict	`array`
`substr(str, inizio, lung)`	Estrae sottostringa	`stringa`

Matrici

Funzione	Descrizione	Ritorno
`matrix(righe, col, val)`	Crea matrice inizializzata	`matrice`
`rows(matrice)`	Numero di righe	`intero`
`cols(matrice)`	Numero di colonne	`intero`
`dim(array)`	Dimensionalità (1 o 2)	`intero`

20. Esempi Pratici

Esempio 1: Calcolatrice

```
// calcolatrice.asc
print("=== CALCOLATRICE ===");
print("Inserisci il primo numero:");
a = to_float(read());

print("Inserisci operatore (+, -, *, /):");
op = read();

print("Inserisci il secondo numero:");
b = to_float(read());

switch (op) {
    case "+": { print("Risultato: " + (a + b)); };
    case "-": { print("Risultato: " + (a - b)); };
    case "*": { print("Risultato: " + (a * b)); };
    case "/": {
        if (b != 0) {
            print("Risultato: " + (a / b));
        } else {
            print("Errore: divisione per zero!");
        }
    };
    default: { print("Operatore non valido"); };
}
```

Esempio 2: Fibonacci

```
// fibonacci.asc
func fibonacci(n) {
    if (n &lt;= 1) {
        return n;
    }
    return fibonacci(n - 1) + fibonacci(n - 2);
}

print("Primi 10 numeri di Fibonacci:");
for (i = 0; i &lt; 10; i += 1) {
    print("F(" + i + ") = " + fibonacci(i));
}
```

Esempio 3: Bubble Sort

```
// bubblesort.asc
arr[0] = 64;
arr[1] = 34;
arr[2] = 25;
arr[3] = 12;
arr[4] = 22;
n = 5;

// Bubble Sort
for (i = 0; i &lt; n - 1; i += 1) {
    for (j = 0; j &lt; n - i - 1; j += 1) {
        if (arr[j] &gt; arr[j + 1]) {
            temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
        }
    }
}

print("Array ordinato:");
for (i = 0; i &lt; n; i += 1) {
    print(arr[i]);
}
```

Esempio 4: Gestione File

```
// file_manager.asc
func salvaNote(filename, contenuto) {
    f = open(filename, "w");
    if (f != NULL) {
        write(f, contenuto);
        close(f);
        return true;
    }
    return false;
}

func leggiNote(filename) {
    f = open(filename, "r");
    if (f != NULL) {
        contenuto = read_all(f);
        close(f);
        return contenuto;
    }
    return "";
}

// Uso
print("Inserisci una nota:");
nota = read();

if (salvaNote("note.txt", nota)) {
    print("Nota salvata!");
    print("Contenuto salvato: " + leggiNote("note.txt"));
} else {
    print("Errore nel salvataggio");
}
```

Esempio 5: GUI Semplice

```
// gui_counter.asc
global contatore = 0;

func incrementa() {
    global contatore = contatore + 1;
    tk_config(label, {text: "Contatore: " + contatore});
}

func decrementa() {
    global contatore = contatore - 1;
    tk_config(label, {text: "Contatore: " + contatore});
}

// Interfaccia
root = tk_root("Contatore");
tk_geometry("300x150");

label = tk_widget(root, "Label", {text: "Contatore: 0", font: "Arial 20"});
tk_pack(label, {pady: 20});

frame = tk_widget(root, "Frame", {});
tk_pack(frame, {});

btnMeno = tk_widget(frame, "Button", {text: " - ", width: 5});
tk_command(btnMeno, "decrementa");
tk_pack(btnMeno, {side: "left", padx: 10});

btnPiu = tk_widget(frame, "Button", {text: " + ", width: 5});
tk_command(btnPiu, "incrementa");
tk_pack(btnPiu, {side: "left", padx: 10});

tk_mainloop();
```

21. Appendice: Riferimento Rapido

Sintassi Essenziale

```
// Variabili
x = 10;
global y = 20;

// Funzioni
func nome(arg1, arg2) { return arg1 + arg2; }

// Struct
struct Nome { campo1, campo2 }
obj = new Nome();

// Controllo flusso
if (cond) { } else if (cond2) { } else { }
for (i = 0; i &lt; n; i += 1) { }
while (cond) { }
switch (val) { case 1: { }; default: { }; }

// Eccezioni
try { } catch (e) { }
throw "errore";

// Include
include "file.asc";
```

Operatori

Aritmetici: + - * / %

Confronto: == != > < >= <=

Logici: && || !

Assegnamento: = += -= *= /=

Valori Speciali

true - valore booleano vero (1)

false - valore booleano falso (0)

NULL - assenza di valore / errore

Ascension v12.6 *- Linguaggio di programmazione per uso didattico e hobbistico*
Autore: EdeFede | Licenza: GPL v3

Ricorda: Ascension È la sintassi. L'implementazione può cambiare, ma l'essenza del linguaggio rimane.