Conceptos de programación

Lenguaje de programación

- Conjunto de instrucciones por medio del cual le expresamos a la computadora lo que debe hacer.
- Sirve para modelar el mundo en un lenguaje intermedio entre el humano y el lenguaje de máquina.
- Ejemplos de lenguajes de programación: Ensamblador, C, C++, Fortran, Java, Python, LISP, IDL, Basic, COBOL, Pascal.

Lenguaje de programación

- Cada lenguaje define un conjunto de ***palabras reservadas*** con las que podemos expresar algoritmos.
- Los lenguajes de programación son independientes del sistema operativo, son una definición abstracta.

Código fuente

- Texto expresado en un lenguaje de programación que compone un programa.
- Generalmente se guarda en archivos de texto y se reconocen por una extensión de acuerdo al lenguaje, ejemplos:
  - Python: .py
  - C, C++ : .c, .cpp
  - Fortran: .f90

Instrucciones

- Las instrucciones son expresiones que usan palabras reservadas del lenguaje y operadores, para indicar una acción simple.
- Generalmente una instrucción es corta, en una sola línea de código.
- En algunos lenguajes, se marca el final de una instrucción con un símbolo especial: en C, es el caracter ***;***

Comentarios

- Un comentario es una línea dentro del código fuente que no se ejecuta.
- Sirve al programador para documentar algo referente al programa sin afectar su ejecución.
- Existen comentarios de línea o de bloque.

```
#Este es un comentario de línea en Python


"""
Este es un comentario de bloque
en Python, puedo poner tantas
lineas como quiera/necesite, no se ejecutarán.
"""
```

```
//Este es un comentario de linea en C

/*Y este es un
comentario de bloque en C que
puede contener cualquier cantidad de lineas*/
```

Bloque de código

- Un bloque de código es un conjunto de líneas/instrucciones agrupadas bajo un contexto común (una función, una condición, etc).
- Nos sirven para ordenar y organizar el código de forma que pueda reutilizarse.
- Cada lenguaje define su propia forma de delimitar bloques de código.
- Los bloques de código pueden contener otros bloques de códigos (anidamiento).

Bloque de código

```c
//Bloques de código de C
int a, b, c, d, e, f, g, h;
while(a < b){
    c = 10;
    d = 100;
    if (c > d){
        e = c;
        f = d;
    }else{
        g = d;
        h = c;
    }
}
```

```python
if row['RA'] == "" and row['DEC'] == "":
    #segmento = {'constelacion':None, 'punto1': None, 'punto2': None}
    continue
else:
    RAc = trunc(float(row['RA']), 2)
    DECc = trunc(float(row['DEC']), 2)
    star_count += 1
    s = match_star({'RA': RAc, 'DE': DECc}, bsc_catalog)
    if s:
        matches += 1
        if not (star_count+1) % 2:   #primer punto
            segmento['constelacion'] = cname
            segmento['punto1'] = s['hrid']
        else:
            segmento['punto2'] = s['hrid']
            const_data.append(segmento)
            segmento = {'constelacion':None, 'punto1': None, 'punto2': None}
    else:
        print 'ALERTA: no match %s'%(i)
```

Variable

- Una variable es una etiqueta que usamos dentro de un programa, para representar/almacenar un valor en memoria.
- Cada lenguaje tiene su forma de declarar variables.
- Algunos lenguajes requieren que la variable se declare antes de ser usada (C, Java, Fortran) otros no (Python).

Declaración de variables

```c
//Declaración de variables en C
int a, b, c, d, e, f, g, h;
float valor1, valor2, valor3;
bool cond1, cond2;
```

```fortran
!Declaración de variables en Fortran
integer*4 N, i, j, itr_inicio, itr_final, myid, n_vecinos, max_i
integer*4 lista_vecinos(0:n_vecinos-1)
integer*4 matriz_vecinos(0:N-1, 0:n_vecinos-1)
```

```python
#Variables in Python
bsc_catalog = {}
const_data = []
path = './data/bsc.dat'
const_path = './data/constellations.csv'
valor1 = 10
valor2 = 665.7
```

Tipos de dato

- Tipo de valores que puede tener una variable:
  - Numéricos (entero, punto flotante)
  - Cadenas de texto
  - Listas
  - Lógicos (verdadero y falso)
- A los anteriores se les conoce como “primitivos”.
- Hay tipos de datos compuestos que forman estructuras complejas agrupando valores primitivos.

Operadores

- Realizan operaciones sobre operandos y devuelven un valor.
  - Matemáticos (+, -, *, /)
  - Lógicos (and, or, not)
  - Comparación
  - Concatenación
  - Asignación

Álgebra booleana – tablas de verdad

X	Y	And	Or
True	True	**True**	**True**
True	False	**False**	**True**
False	True	**False**	**True**
False	False	**False**	**False**

X	Not X
True	**False**
False	**True**

Ejecutable

- Un ejecutable es un programa o script cuyas instrucciones se envían al procesador para ser *ejecutadas*.
- En linux, no importa la extensión, solo debe tener permisos de ejecución (chmod +x).
- Puede ser un archivo binario creado a partir del código fuente.
- Puede ser un script procesado por una máquina virtual (java) o intérprete (python).

Compilador

- Cada lenguaje de programación tiene un compilador (o intérprete).
- Es un programa que toma un código fuente, lo valida, y lo convierte a código ejecutable.
- Ejemplos de programas compiladores:
  - C: gcc
  - Fortran: gfortran
  - Java: javac
  - Python: python

Validación del compilador

```
  GNU nano 2.2.6                File: prueba.c

#include <stdio.h>

int main(){
        int a = 10;
        int c = 15
        printf("HOLA\n");
        return 0;
}
```

```
  GNU nano 2.2.6                File: prueba.py

valor1 = 10
valor2 = 20
valor3 = valor1 + valor4
print("HOLA: %d"%(valor3))
```

```
File  Edit  View  Search  Terminal  Help
jeudy@machine:~/Proyectos/UCR$ gcc prueba.c
prueba.c: In function ‘main’:
prueba.c:6:2: error: expected ‘,’ or ‘;’ before ‘printf’
jeudy@machine:~/Proyectos/UCR$
```

```
jeudy@machine:~/Proyectos/UCR$ python prueba.py
Traceback (most recent call last):
  File "prueba.py", line 3, in <module>
    valor3 = valor1 + valor4
NameError: name 'valor4' is not defined
jeudy@machine:~/Proyectos/UCR$
```

Lenguajes compilados VS interpretados

- Compilados: compilador transforma código fuente en un binario con instrucciones para el procesador. Ejs: C, C++, Fortran.
- Interpretado: compilador transforma código fuente a un estado intermedio procesado por una máquina virtual, quien es la que lo ejecuta. Ejs: Python, Java.

Lenguajes compilados VS interpretados

- En java, al código intermedio se le conoce como “Byte codes” y es ejecutado por la Java Virtual Machine.
- Los lenguajes compilados producen *por lo general*, programas más rápidos.
- Los lenguajes interpretados como python, son más flexibles y mas interactivos.
- Más info y discusión:
  http://stackoverflow.com/questions/441824/java-virtual-machine-vs-python-interpreter-parlance

Algoritmo

- Secuencia de pasos para resolver un problema.
- Ejemplos: cálculo de raíz, ordenar una lista, cálculo de una integral, cálculo de Pi.
- Órden de un algoritmo: una medida de que tan eficiente es un algoritmo para resolver un problema basado en la entrada.
- Un mismo problema puede ser resuelto por diferentes algoritmos de diferente órden.

Órden de un algoritmo

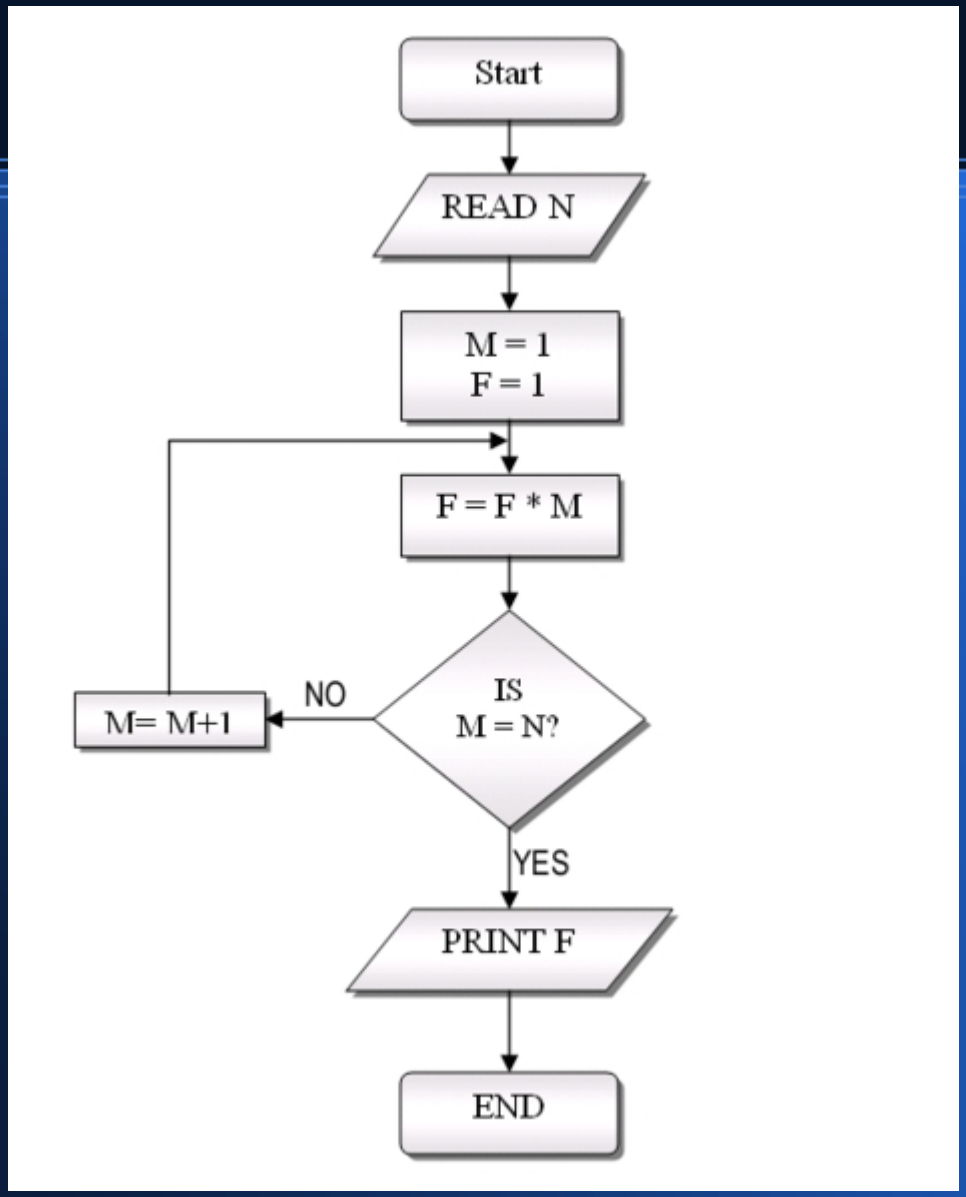
- O(1): constante. Sin importar tamaño de entrada, siempre ejecuta mismo número de operaciones. Ejemplo: determinar si un número es par o impar.
- O(n): linear. La duración depende linearmente de número de elementos. Ej. 1000 elementos de entrada, 1000 operaciones.

Órden de un algoritmo

- O(log n): logaritmico. Un poco menos eficiente que linear.
- $O(n^2)$: cuadrático: 100 elementos, 100000 operaciones.
- Métodos de fuerza bruta son por lo general $O(n^2)$ y se busca optimizarlos a logarítmico o lineal.

Diagramas de Flujo

- Representación gráfica de un algoritmo.
- Por lo general se trabaja primero en el diagrama de flujo antes de programar para asegurarse de que se entiende el problema y el algoritmo funciona.
- Define la dirección de la ejecución del programa según se cumplen determinadas condiciones.


Start
READ N
M = 1
F = 1
F = F * M
IS
M = N?
NO
M= M+1
YES
PRINT F
END

Control de flujo

- Si se cumple una condición (compuesta por expresiones, variables y operadores) se entra a un determinado bloque de código.
- Sino se cumple, se entra a otro bloque de código.
- La evaluación de condiciones puede ser anidada.

Control de flujo

- Cada lenguaje tiene sus propias estructuras de control de flujo, pero en general se usan ***if*** y ***else***.

```
if s:
    matches += 1
    if not (star_count+1) % 2:
        segmento['constelacion'] = cname
        segmento['punto1'] = s['hrid']
    else:
        segmento['punto2'] = s['hrid']
        const_data.append(segmento)
        segmento = {'constelacion':None, 'punto1': None, 'punto2': None}
else:
    print 'ALERTA: no match %s'%(i)
```

Ciclos

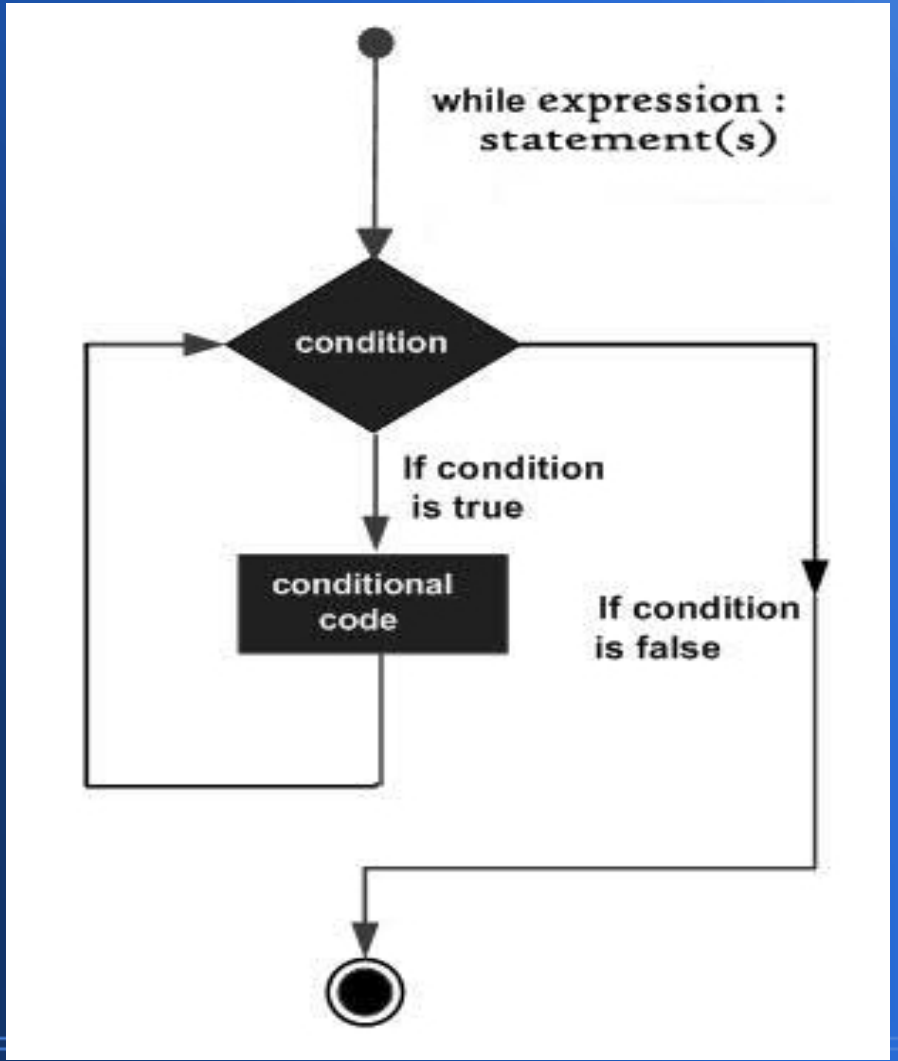
- Bloques de código que se ejecutan repetidamente mientras que se cumpla una condición (condición de parada).
- A cada ejecución del bloque de código se le conoce como iteración.
- La condición de parada puede ser compleja y estar formada por variables, operadores, llamadas a funciones, etc.
- En python, hay ciclos para recorrer estructuras como listas.

Ejemplo de ciclo en python

```
#!/usr/bin/python

count = 0
while (count < 9):
   print 'The count is:', count
   count = count + 1

print "Good bye!"
```



Ciclos infinitos

- Los ciclos deben usarse con cuidado. Puede caerse en ciclos infinitos.
- Hay que asegurarse de que la condición de parada se cumple eventualmente.

```
#!/usr/bin/python

var = 1
while var == 1 :  # This constructs an infinite loop
   num = raw_input("Enter a number  :")
   print "You entered: ", num
```

Funciones

- Una función, es un conjunto de instrucciones a la que se le da un nombre.
- La ventaja de las funciones es que permiten reutilizar el código.
- En lugar de estar llamando el mismo bloque de código muchas veces, se define una función y se llama.

Funciones

- Por lo general, las funciones devuelven un valor que puede capturarse y asignarse a una variable, o usarse dentro de expresiones complejas con operadores.
- Las funciones pueden recibir parámetros al invocarse.
- Cada lenguaje tiene su propia sintaxis.

Ejemplo de función en Python

```
Definición de función para elevar
un número X a la N potencia
"""

def elevarXN(x, n):
    if n == 0:
        return 1
    else:
        resultado = 1
        while n > 0:
            resultado = resultado * x
            n = n - 1
        return resultado

#Ejemplo de llamadas de la función elevar

resultado1 = elevarXN(2, 3)
print resultado1

resultado2 = elevarXN(10, 3)
print resultado2
```

Modulos/Librerias

- Grupos de funciones con un tema/área en común.
- Se separan como un componente aislado que puede utilizarse desde cualquier otro programa.
- Cada lenguaje define como se crean e importan módulos.
- Python permite importar módulos enteros o bien funciones específicas dentro de estos.

Debugger

- El Debugger es una herramienta que nos permite ejecutar un programa instrucción por instrucción.
- Su utilidad es para depurar un programa, encontrar errores, ver resultados parciales, caminos tomados en el flujo de ejecución.
- Python tiene su propio debugger, y puede integrarse con herramientas gráficas de programación.