4. Instruir el discurso: Asignaciones

Empezamos a continuación con lo que se podría denominar la parte nuclear de la asignatura, que comprende básicamente los capítulos 4-8, en donde **diseñaremos ya nuestros primeros programas** totalmente funcionales en los distintos lenguajes de programación. Para ello, usaremos a lo largo de este curso varias herramientas de software para los varios lenguajes de programación ya mencionados.

No obstante, cabe destacar que este libro no utiliza la metodología clásica en la que se enseña el lenguaje de programación como un idioma del que se debe obtener un título de nivel desde A1 a C2. Personalmente, esto lo considero un error muy grave, ya que merma la capacidad de razonar del futuro programador, pues se tiende a mecanizar los problemas, ya que en la metodología clásica, se van enseñando poco a poco las sentencias que se pueden dictar con el lenguaje de programación, para qué sirven y cómo aplicarlas, desde conceptos básicos casi comunes hasta conceptos muy nativos pero qué sólo funcionan en ese lenguaje.

El objetivo de esta obra es poder acercarnos a hablar con del ordenador, no tanto su idioma, sino acercándonos a **cómo piensa el ordenador**; por ello el subtítulo no dice "guía para hablar", sino "guía para discernir". Los lenguajes de programación (idiomas) son en realidad implementaciones de los algoritmos (ideas) que hemos estado diseñando a lo largo de esta obra, y el **pseudocódigo** y el **diagrama de flujo** son como dos lenguajes universales comprensibles a nivel humano alrededor de todo el globo terráqueo y que **simulan el pensamiento computacional**, pero **el ordenador no es capaz de comprenderlo**.

La **Programación** es uno de los **tópicos más importantes de la Ingeniería Informática**, así como la **base mínima** que en este siglo de la **digitalización** todo ingeniero debe tener, por no decir que es la llave que abre las puertas entre el mundo exterior y el ordenador. Todo uso que se te ocurra de las Nuevas Tecnologías, en especial de las TICs, o las **Tecnologías de la Información y de la Comunicación**, necesita programas.

Para poder resolver estos grandes problemas, no se consta ni de mecanizar ni de copiar y pegar métodos de los grandes programadores, **la Programación consiste en resolver a través del razonamiento de un ordenador y de forma razonada un problema**, siguiendo las normas gramaticales de un lenguaje determinado y explicar por qué se ha resuelto en esa serie de pasos determinado. De nada sirve talar un bosque si no se sabe hacer leña. Y es que al fin de al cabo, todos los caminos llevan a Roma, puede existir más de una resolución al problema. Cualquier problema que se nos ocurra, lo puede resolver nuestro ordenador siempre y cuando sepamos qué hay que hacer y se lo indiquemos en una forma que lo comprenda, como a un gólem cuando le introducimos un papel.

Asentemos a continuación nuestras bases. ¿Cómo podemos hacer un programa totalmente funcional y que lo comprenda el ordenador?

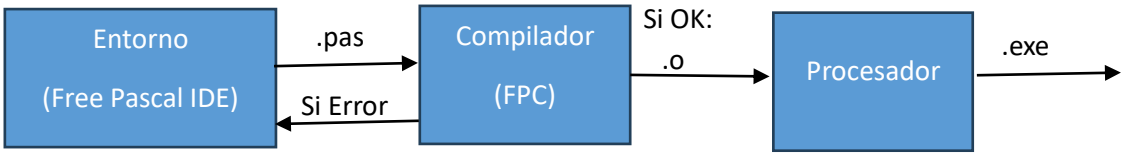
En primer lugar, debemos conocer en qué ambientes puede un ordenador ser capaz de reconocer un programa escrito en un lenguaje y procesarlo. Hablamos de los **compiladores** y los **entornos de programación**.

Los entornos son **aplicaciones idóneas donde podremos crear archivos con una extensión la cual el compilador puede leer su contenido** (que es nuestro programa codificado en ese

determinado lenguaje de programación) **y traducir las instrucciones de dicho programa al lenguaje máquina** para que el procesador de nuestro ordenador ejecute la tarea encomendada.

Cabe diferenciar entre entorno de programación, que es donde se **crea el archivo de nuestro programa** y se envía al **compilador**; y este último, que es lo que **lee nuestro archivo** y **transforma** el conjunto de instrucciones de nuestro lenguaje de programación **al lenguaje máquina** y lo envía al procesador. Muchas veces, los compiladores no se incluyen en la instalación del entorno, y deben instalarse aparte. Por ejemplo, para ADA debemos instalarnos el compilador GNAT y el entorno AdaGide, y para Python y C debemos descargarlos dentro de la propia aplicación de VS Code.

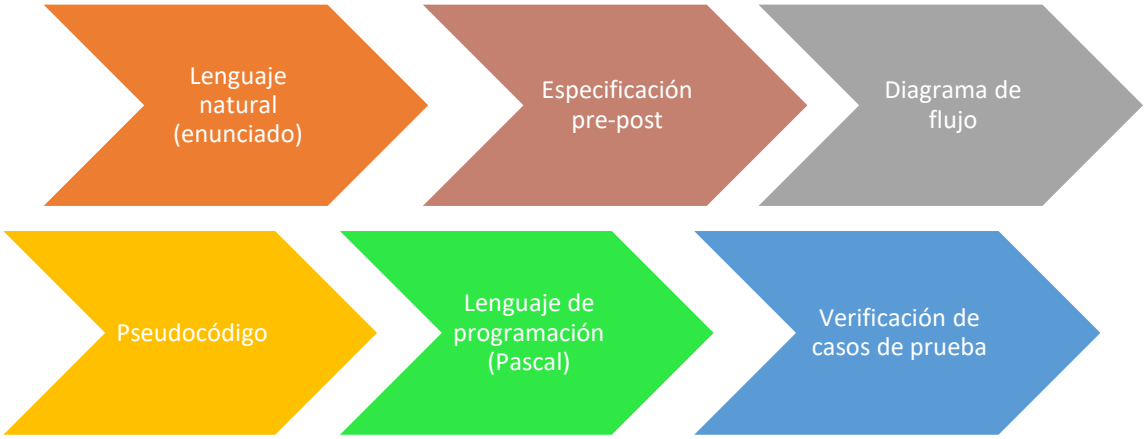
Nosotros usaremos a partir de ahora por defecto el lenguaje Pascal. Para programar en este lenguaje, basta con descargar Free Pascal, que incluye ambos, entorno de programación (Free Pascal IDE) y compilador (FPC).



En este entorno, codificaremos nuestros programas en el lenguaje Pascal, y los guardaremos en la extensión propia del lenguaje, pas. Además, este entorno permite ejecutar nuestros programas. Para ello, el entorno envía nuestro archivo pas al compilador. Si todo es correcto, el compilador crea nuestro código máquina de extensión o y lo envía al procesador, que ejecutará el programa, creando un ejecutable (.exe). En caso contrario, el compilador enviará un mensaje de error y la línea del documento en la que se ha detectado.

A modo de anexo, se adjuntarán guías breves para programar en los cuatro lenguajes que usaremos en este curso: Pascal (por defecto), ADA, C y Python.

Vamos a diseñar nuestro primer sencillo programa, será un programa en el que dado el radio de un círculo, calcule el área de este. Su fórmula es: $A = \pi r^2$, siendo el valor de π (pi) aproximadamente 3.1416. Como principiantes en la programación, se sugiere seguir la siguiente serie de **pasos para** poder **diseñar cualquier** tipo de **programa**:



1. **Leer y comprender el enunciado**, qué se pide, qué datos de entrada tenemos y qué datos de salida se esperan. Detectar también posibles ambigüedades.
2. Realizar la **especificación pre-post** del programa, los tipos de datos esperados, qué condiciones se esperan que cumplan los datos de entrada y los datos de salida.
3. Diseñar un buen **diagrama de flujo** ayuda a visualizar mejor las **ideas principales** del algoritmo. Establece la serie de pasos que hay que realizar para ejecutar el problema en un

borrador u hoja en sucio y después expresa esos pasos en bloques y sus transiciones en líneas de flujo.

4. Observando las **estructuras de control** que contiene el diagrama de flujo, podemos fácilmente intuir el **algoritmo en pseudocódigo**. En el pseudocódigo, podemos utilizar todas las herramientas que se nos ocurran, independientemente de si estas son soportadas por el lenguaje solicitado o no.
5. Finalmente, **desarrollamos el programa** cumplimentando con la estructura similar al texto instructivo que hemos visto: cabecera del título, especificación, declaraciones e **implementación del algoritmo en ese lenguaje de programación**. En este paso, nuestras herramientas son limitadas y **debemos ajustarnos a las reglas gramaticales** del lenguaje.
6. No viene mal agrupar en una tabla una serie de **casos de prueba significativos** y sus resultados esperados y **ejecutar el** nuevo **programa** confiriendo esos valores a los datos de entrada para comprobar que **se obtienen los resultados esperados**.

Esto es sólo una **metodología básica e inicial** que se sugiere usar con el fin de ayudar al joven programador a diseñar sus programas correctamente, especialmente aquellos más complejos. En cuanto avancemos más, podremos irnos más directos al grano, al paso 5, pero al menos en esta parte nuclear de la asignatura es altamente recomendable seguir estos pasos, sobre todo si eres nuevo en el mundo de la programación.

Recordamos que un programa es correcto cuando **coincide lo que se obtiene de valor salida al ejecutar el programa y lo que se debería obtener según la especificación** del programa, por ello es muy importante hacer hincapié en realizar buenas especificaciones y en procurar seguir el diseño del programa según ese contrato. Un programa que no sigue su especificación no es correcto.

Además, recuerda que todos los caminos llevan a Roma, por lo que **dos programas distintos pueden resolver el mismo problema correctamente**, y por tanto los dos son iguales de válidos. Su diferencia radica simplemente en la sencillez con la que se puede seguir el diseño del programa (longitud de las instrucciones, complejidad de las estructuras de control, nº de variables a declarar, etc.) o eficiencia. Entre todas las características de los programas, **Bases de la Programación priorizará la corrección**.

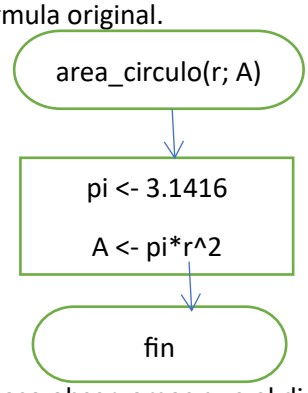
A continuación, procedemos a diseñar el programa propuesto del área del círculo.

El área de círculo, que es lo que queremos obtener, recibe como única incógnita el radio del círculo, único dato de entrada. El enunciado no es ambiguo, pues aunque no se ofrece la unidad de medida del radio, la unidad que el usuario decidiera tomar no afectaría en absoluto al resultado del área.

El tipo de dato radio puede ser o bien entero o bien real, pero asumiremos que r es real porque es el tipo de dato menos restringido de ambos. La especificación pre-post del programa es la siguiente:

- Datos de entrada: un real r, radio de la circunferencia
- Precondición: r > 0.0
- Datos de salida: el área del círculo, A, nº real
- Postcondición: A = 3.1416 * r^2

Cabe recalcar que **si no se menciona nada sobre entrada por teclado o salida en pantalla, los datos de entrada y salida se obtienen desde memoria**, y será así hasta dar el tema 7. Procedemos a continuación a diseñar su respectivo diagrama de flujo y posteriormente, el pseudocódigo. En este caso se trata de un algoritmo muy sencillo, ya que tan sólo hay que aplicar una fórmula matemática. No obstante, para optimizar mejor el código, vamos a declarar una variable PI <- 3.1416, siendo más fieles a la fórmula original.




```
area_circulo(r; A){
  pi := 3.1416;
  A := pi * r^2;
}
{ En pseudocódigo, deben indicarse los datos
de E/S estándar o de memoria que se toman }
```

En este caso observamos que el diagrama de flujo sigue una **estructura secuencial**, lo que quiere decir que **ejecuta las instrucciones una a una, en orden y obligatoriamente**. Vamos a ver a continuación cómo desarrollar un programa en estructura secuencial.

En primer lugar, dos cosas comunes a todos los programas son el **título del programa y la especificación** de este. El título debe tener un nombre que al leerlo nos dé una idea de para qué sirve el programa. La especificación preferiblemente debe ser la pre-post, y debe introducirse a modo de comentario. Ambas partes del programa dependen por supuesto del lenguaje de programación que utilicemos.

En Pascal, la cabecera del programa debe ir precedida de la palabra reservada program, y sucedida por un punto y coma (;) y los comentarios se introducen entre dos llaves. El inicio de nuestro programa en Pascal debería ser el siguiente:

```
program area_circulo;

{ Datos de entrada: un real r, radio de la circunferencia
Precondición: r >0.0
Datos de salida: el área del círculo, A, nº real
Postcondición: r = 3.1416 * r^2 }
```

Posteriormente, debemos **declarar toda variable o tipo no primitivo** que utilicemos, lo que nos abre una nueva sección en este tema. Estas declaraciones actúan como si fueran una lista del **inventario** que hay que tener a mano para seguir una receta o una tarea.

4.1 Declaraciones

En los lenguajes de programación **de alto nivel**, cuando se quiera utilizar una variable, **es obligatorio indicar anteriormente el tipo de dato que se va a usar con ella**, como si fuera un listado de las cosas que hay que tener antes de ejecutar una tarea. Como se ha observado **en el pseudocódigo**, a diferencia de en el programa, **no es obligatorio declarar variables**.

La manera en la que se realizan las declaraciones vuelve a depender por supuesto del lenguaje de programación. Además, cabe distinguir entre constantes y variables.

Las constantes se declaran al principio del programa y, como su nombre indica, tienen la peculiaridad de que, **una vez declarado su valor, lo mantienen durante toda la ejecución** del algoritmo. Si el algoritmo intentara por lo que fuera manipular el valor de una constante (p.ej. se intenta ejecutar la instrucción pi:=2*pi, siendo pi=3.1416 declarada como constante), daría lugar a un error de compilación; lo que le confiere mayor seguridad a la hora de programar. **La inicialización del valor de una constante es obligatoria**.

Por otro lado, **las variables pueden cambiar de valor a lo largo de la ejecución del código**, de ahí su nombre. Su inicialización no es obligatoria, pero si se ejecutara una instrucción en donde se necesitara su valor preciso y esta no está o bien inicializada o bien asignada (p.ej. declarar una variable x y se ejecuta la instrucción y := x + 1 sin conferir anteriormente un valor a x), se producirá un error de compilación.

Evidentemente, también dará un error de compilación si ejecutamos una instrucción que utiliza una variable no declarada.

Por lo que, siguiendo las reglas de Pascal, las declaraciones se realizan antes del algoritmo, y las constantes y las variables se declaran separadas en dos secciones, una sucedida por la palabra reservada const y otra por la palabra var, siguiendo la siguiente sintaxis:

<declaración_constantes> ::= <nombre_constante> : <tipo_dato> := <valor_constante> ;

<declaración_variables> ::= <nombre_variable> : <tipo_dato> [:= <valor_inicio>] ;

```
const
   PI : Real := 3.1416;
var
   A, r: Real;
```

Cabe destacar que **por convenio, las constantes se escriben en mayúsculas mientras que las variables se escriben en minúsculas**. Este convenio se establece porque en lenguajes de bajo nivel, como son débilmente tipados y por ende no se realizan declaraciones, los desarrolladores distinguen así entre lo que está diseñado para que sea constante y lo que está diseñado para que sea variable. Sin embargo, los lenguajes de bajo nivel como Python no logran evitar que a lo largo del programa se pueda modificar el valor de una constante, ya que son tratadas como variables igualmente.

Por otra parte, otros lenguajes como C no distinguen entre la parte donde se escriben las declaraciones de la parte donde se escriben las asignaciones y resto de instrucciones.

Aparte de constantes y variables, todo tipo de dato no primitivo del lenguaje también debe declararse. Dependiendo del tipo de dato, la manera de declararlos también variará, pues no es lo mismo una enumeración que una rodaja, por ejemplo.

En Pascal se procedería siguiendo esta sintaxis, sucedida de la palabra reservada type:

<declaración_enumeración> ::= <nombre_tipo> = (<tag1>, … , <tagn>);

<declaración_rodaja> ::= <nombre_tipo> = <valor_inicio>..<valor_fin>;

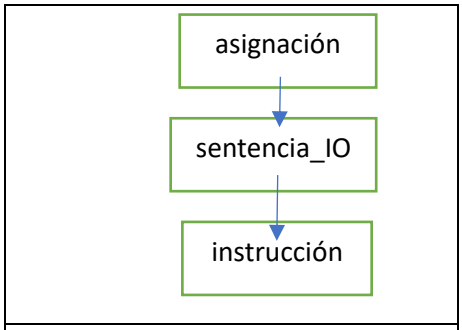
Así por ejemplo, para definir en un programa de Pascal la enumeración de días de la semana y la rodaja de nº positivos, se procedería de la siguiente manera:

```
type
   dias = (L, M, X, J, V, S, D);
   natural = 0..MaxInt;
```

Como siempre, esto depende bastante del lenguaje de programación, y de hecho, en C no se aceptan rodajas y en Python no se aceptan ninguno de los dos TDDs vistos por ahora.

4.2 Estructura secuencial

La estructura secuencial se refiere a un diseño del algoritmo en el cual **una serie de instrucciones son ejecutadas en el orden en el que aparecen**, de arriba a abajo.




```
<asignación>::= <variable> := <valor> | <expresión> ;
<sentencia_IO>::= scan() | print() ;
<instrucción>::= <asignación> | <sentencia_IO>;
```

Estructura secuencial.
Es aquella que ejecuta una sucesión de instrucciones secuencialmente, ejecutándolas todas y sólo una vez.

La estructura secuencial tiene las siguientes características:

- Sus instrucciones se siguen de forma que **el valor de salida que se obtiene tras ejecutar una instrucción es utilizado como valor de entrada para ejecutar su siguiente** instrucción
- El programa **sólo** tiene **un punto de entrada y sólo un punto de salida**
- La **ejecución** de las instrucciones es **unidireccional** (de arriba abajo, sin retroceder nunca), **lineal** (una a una y en orden) **y de una única vez**.
- La lectura de las instrucciones es sencilla, y por lo tanto **fácil de mantener**, ya que sólo se emplean **asignaciones y operaciones básicas** (entre ellas operaciones características de cada tipo de dato y operaciones de E/S que veremos en el tema 7)

En el caso del programa del área del círculo, observamos que se trata de un programa que se puede resolver mediante únicamente asignaciones, que actúa como función matemática (y por lo tanto, un único punto de entrada y un único punto de salida), que sólo es necesario ejecutar una sola vez un número de instrucciones, que tienen un orden concreto y que además, el resultado de la primera determina el resultado de la segunda. Por todo ello, podemos inferir que la estructura más adecuada para diseñar el algoritmo de este ejemplo es la estructura secuencial.

Siguiendo la sintaxis de Pascal de las asignaciones, que es la siguiente:

```
<asignación>::= <variable> := <valor> | <expresión> ;
```

Tendríamos únicamente una instrucción a ejecutar, ya que la instrucción de pseudocódigo en donde se había asignado a pi el valor 3.1416, se sustituye por la declaración de la constante. Esta instrucción corresponde con la ejecución de la fórmula matemática $A = \pi r^2$

```
A := PI*r*r;
```

Hay que tener en cuenta que la operación exponenciación (como la que se presenta, r^2), no existe en Pascal, por lo que debemos pensar en otras formas de ejecutar dicha operación. Sabemos que por definición: $x^n = x \cdot x \cdot x \cdot x \ldots$ n veces, luego, $x^2 = x \cdot x$.

La secuencia de instrucciones de nuestro programa, en Pascal, debe ir comprendida entre las palabras reservadas begin y end. El programa en Pascal quedaría finalmente de la siguiente forma:

```
program area_circulo;
{$Assertions ON}
{ Datos de entrada: un real r, radio de la circunferencia
Precondición: r >0.0
Datos de salida: el área del círculo, A, nº real
Postcondición: r = PI * r^2, con PI = 3.1416}

const
   PI : Real := 3.1416;
var
   A, r: Real;
begin
   r := 1; {Inicializar datos de entrada}

   A := PI*r*r;

   Assert(A = PI); {Verificar caso de prueba}

end.
```

No obstante, a la hora de programar nuestros primeros programas, debemos tener en cuenta tres aspectos:

- Estos programas **no reciben entrada por teclado ni imprimen en pantalla**
- Por lo dicho anteriormente, esto causará un error de compilación porque los datos de entrada que no han sido inicializados tienen que procesarse en una asignación posterior. Por ello, es **obligatorio inicializar en el código los datos de entrada**, preferiblemente con los valores de un **caso de prueba** significativo
- No estaría de más **comprobar el correcto funcionamiento del programa**. Para ello, utilizamos la funcionalidad **Assert**. Para habilitarla, se ha de escribir tras la cabecera del programa `{$Assertions ON}`. La sintaxis de esta funcionalidad es: `<aserción>::= Assert( <dato_de_salida> = <valor_esperado> )` y con esto, **comprobamos que dados los datos de entrada del caso de prueba, obtengamos los datos de salida esperados**.

  Si la aserción es verdadera, el programa termina silenciosamente, ya que ni recibe ni imprime datos en pantalla (no hay comunicación alguna con el usuario), lo que significa que **el programa es parcialmente correcto y** que **el caso de prueba se ha verificado** correctamente. **En caso contrario, el compilador lanzará un error de aserción** el cual será visible desde nuestro entorno de programación, por lo que **el programa no es correcto**.