1. Introducción a la programación en Python

[1]:
```
import funciones_varias  as fv
```

1 Características del lenguaje Python

- Legible
- Entorno de desarrollo interactivo
- Compacto y expresivo (lenguaje de muy alto nivel=)
- ***Case Sensitive*** (sensible al uso de mayúsculas y minúsculas)

Ejemplo de uso de lenguaje C vs Python

```
/* Promedio de valores utilizando lenguaje C */
#include <stdio.h>

int main(int argc, char* argv[])
{
    float a, b, promedio;
    printf("Escribe un valor: ");
    scanf("%f", &a);
    printf("Escribe otro valor: ");
    scanf("%f", &b);
    promedio=(a+b)/2.0;
    printf("Valor medio = %f\n", promedio);
return 0;
}
```

[2]:
```
''' Promedio de valores utilizando lenguaje Python '''
a = float(input('Escribe un valor: ')) # Lee un valor por teclado y lo convierte␣
 ↪a real
b = float(input('Escribe otro valor: '))
promedio = (a+b)/2.0
print ('Valor medio = ', promedio)
```

```
Escribe un valor:  3
Escribe otro valor:  7
```

```
Valor medio =  5.0
```

Comentarios en lenguaje Python:

- Cualquier texto delimitado en sus extremos por tres caracteres ' o bien " es un comentario
- El carácter # indica que el resto de la línea es un comentario

```
''' ********************************
     Ejemplo de comentario que ocupa
            una o más líneas
    ********************************'''

""" Forma alternativa para delimitar comentarios """

a=3 # Comentario de una línea
```

2 Tipos de datos

- Caracteres: **Código ASCII**
- Números naturales: **Binario natural** (base 2)
- Números enteros: **Complemento 2**
- Números reales: Estándar **IEEE 754**
- Valores Lógicos: ***True*** y ***False***

2.1 Tabla de Caracteres ASCII

https://www.ascii-code.com

ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol
0	0	NUL	16	10	DLE	32	20	(space)	48	30	0	64	40	@	80	50	P	96	60	`	112	70	p
1	1	SOH	17	11	DC1	33	21	!	49	31	1	65	41	A	81	51	Q	97	61	a	113	71	q
2	2	STX	18	12	DC2	34	22	"	50	32	2	66	42	B	82	52	R	98	62	b	114	72	r
3	3	ETX	19	13	DC3	35	23	#	51	33	3	67	43	C	83	53	S	99	63	c	115	73	s
4	4	EOT	20	14	DC4	36	24	$	52	34	4	68	44	D	84	54	T	100	64	d	116	74	t
5	5	ENQ	21	15	NAK	37	25	%	53	35	5	69	45	E	85	55	U	101	65	e	117	75	u
6	6	ACK	22	16	SYN	38	26	&	54	36	6	70	46	F	86	56	V	102	66	f	118	76	v
7	7	BEL	23	17	ETB	39	27	'	55	37	7	71	47	G	87	57	W	103	67	g	119	77	w
8	8	BS	24	18	CAN	40	28	(	56	38	8	72	48	H	88	58	X	104	68	h	120	78	x
9	9	TAB	25	19	EM	41	29	)	57	39	9	73	49	I	89	59	Y	105	69	i	121	79	y
10	A	LF	26	1A	SUB	42	2A	*	58	3A	:	74	4A	J	90	5A	Z	106	6A	j	122	7A	z
11	B	VT	27	1B	ESC	43	2B	+	59	3B	;	75	4B	K	91	5B	[	107	6B	k	123	7B	{
12	C	FF	28	1C	FS	44	2C	,	60	3C	<	76	4C	L	92	5C	\	108	6C	l	124	7C	|
13	D	CR	29	1D	GS	45	2D	-	61	3D	=	77	4D	M	93	5D	]	109	6D	m	125	7D	}
14	E	SO	30	1E	RS	46	2E	.	62	3E	>	78	4E	N	94	5E	^	110	6E	n	126	7E	~
15	F	SI	31	1F	US	47	2F	/	63	3F	?	79	4F	O	95	5F	_	111	6F	o	127	7F	

2.2 Variables

- Para poder conservar y reutilizar los resultados de las operaciones, los datos se almacenan en variables
- Una variable está almacenada en una zona de la memoria del sistema seleccionada por el sistema operativo
- En un programa, una variable se identifica por su nombre

- La asignación de valor a una variable se consigue utilizando el signo =:

```
identificador_de_variable = valor
```

Ejemplos de asignación de valores a variables:

```
letra = 'A'
numero_entero = 1
numero_entero_formato_hexadecimal = 0xFF0A
numero_real = 5.4
numero_real_2 = -3.5
valor_logico = true
cadena_de_caracteres = 'Esto es una cadena de caracteres (string)'
cadena_2 = "Esta es otra cadena de caracteres"
```

El **valor** asignado a una variable **puede ser el resultado de ejecutar una expresión**.

Identificadores válidos:

- Combinaciones de letras minúsculas, mayúsculas, dígitos, y el carácter _ (*underscore*)
- No se permite el uso de las palabras reservadas del lenguaje Python:

```
and, assert, break, class, continue, def, del, elif, else, except, exec, finally, for,
from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try,
while, yield, True, False
```

3 Operaciones y Precedencia

Operador	Funcionalidad
( )	Paréntesis
**	Exponenciación
~, +, -	Complemento a 1, más y menos unarios
*, /, %, //	Multiplicación, división, módulo (resto de la división entera) y cociente de la división entera
+, -	Suma y resta (complemento a 2)
>>, <<	Desplazamiento de bits a la derecha y a la izquierda
&	Operación AND (entre bits)
^, |	Operaciones OR exclusiva y OR (entre bits)
<=, <, >, >=	Operadores de comparación
==, !=	Operadores igualdad y desigualdad
=, %=, /=, //=, -=, +=, *=, **=	Operadores de asignación
is, is not	Operadores de identidad
in, not in	Operadores de pertenencia (membresía)
not, and, or	Operadores lógicos (orden de precedencia:not>and>or)

Ejemplos de operaciones y precedencia

Operaciones básicas

Utiliza los valores a=5 y b=3 para realizar las operaciones que se indican:

[52]:
```
# Valores de a y b:
a=5
b=3
```

a^b

[53]:
```
a**b
```

[53]: 125

Complemento-1 de a (almacena el resultado en la variable `result`)

[54]:
```
result = ~a
fv.print_int_bits(a,num_bits=8)
```

```
fv.print_int_bits(result,num_bits=8)
```

```
05h :: 00000101b :: 5
-6h :: 11111010b :: 250
```

(Complemento-1 de a) + 1

[55]:
```
result = ~a+1
fv.print_int_bits(result,num_bits=8)
```

```
-5h :: 11111011b :: 251
```

Opuesto de a (comprueba que su valor coincide con el resultado anterior)

[56]:
```
-a
```

[56]:
```
-5
```

Cociente de la división entera a/b (el resultado es un número entero)

[57]:
```
a//b
```

[57]:
```
1
```

Resto de la división entera a/b

[58]:
```
a%b
```

[58]:
```
2
```

Resultado real de dividir a entre b

[59]:
```
a/b
```

[59]:
```
1.6666666666666667
```

$(a-b)^{\text{módulo}(b/a)}$

[60]:
```
(a-b)**(b%a)
```

[60]:
```
8
```

Operaciones con **números complejos** `a+bj`

- El coeficiente de la parte imaginaria ha de explicitarse para que $j = \sqrt{-1}$

[61]:
```
complex_a=8.2+2j
complex_b=complex(4.2,-1)  #Equivale a 4.2-1j
print(complex_a,complex_b)
```

```
(8.2+2j) (4.2-1j)
```

[62]:
```
complex_c=complex_a*1j
print (complex_c)
```

```
(-2+8.2j)
```

[63]:
```
complex_d=complex_a-complex_b
print(complex_d)
```

```
(3.999999999999999+3j)
```

[64]:
```
complex_e= 5-2j
complex_f= 1+1j
complex_g=complex_e/complex_f
print (complex_g)
```

```
(1.5-3.5j)
```

Desplazamiento de bits

Resumen:

- `valor >> n` -> Desplaza los bits de la variable valor n posiciones hacia la derecha
- `valor << n` -> Desplaza los bits de la variable valor n posiciones hacia la izquierda

Crea la variable `dato = 0000 1100b = 0Ch`

[65]:
```
dato=0x0C # inicialización con valor hexadecimal
fv.print_int_bits(dato)
```

```
000Ch :: 0000000000001100b :: 12
```

Calcula el resultado de desplazar dato 3 bits a la izquierda y guarda el resultado en la propia variable dato

(Comprueba que `dato<<` es una forma, rápida, de calcular el valor dato $\cdot 2^n$)

[66]:
```
dato=dato<<3
fv.print_int_bits(dato)
```

```
0060h :: 0000000001100000b :: 96
```

Actualiza la variable dato desplazándola 1 bit hacia la derecha

(Comprueba que `dato>>` es una forma, rápida, de calcular la división entera $\frac{\text{dato}}{2^n}$)

[67]:
```
dato=dato>>1
fv.print_int_bits(dato)
```

```
0030h :: 0000000000110000b :: 48
```

Actualiza dato con un desplazamiento a la derecha de 5 bits

(Observa que se pierden bits al realizar el desplazamiento)

[68]:
```
dato=dato>>5
fv.print_int_bits(dato)
```

```
0001h :: 0000000000000001b :: 1
```

AND, OR y OR-eXclusiva a nivel de bit

Resumen:

- Tablas de verdad:

bit a	bit b	a AND b	a OR b	a XOR b
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

- Las tablas de verdad se pueden interpretar de la siguiente forma:

bit a	bit_b	a AND b	a OR b	a XORb
0	X	0	X	X
1	X	X	1	~X

Crea las variables `signal = 1001 0110 b = 96h` y `mask = 0000 1111 b = 0Fh`

[69]:
```
signal=0x96
mask=0x0F
```

Calcula el valor signal AND mask

(Comprueba que el resultado conserva los bits de signal en las posiciones en que mask tiene valor 1, mientras que se tienen 0s en las posiciones en las que mask es 0)

[70]:
```
resultado = signal & mask

fv.print_int_bits(signal,formato='bin',num_bits=8)
fv.print_int_bits(mask,formato='bin',num_bits=8)
fv.print_int_bits(resultado,formato='bin',num_bits=8)
```

```
10010110b :: 150
00001111b :: 15
00000110b :: 6
```

Calcula el valor signal OR mask

(Comprueba que los bits de signal se conservan en las posiciones de valor 0 en mask, mientras que se ponen a 1 en las posiciones con valor 1 en mask)

[71]:
```
resultado = signal | mask

fv.print_int_bits(signal,formato='bin',num_bits=8)
fv.print_int_bits(mask,formato='bin',num_bits=8)
fv.print_int_bits(resultado,formato='bin',num_bits=8)
```

```
10010110b :: 150
00001111b :: 15
10011111b :: 159
```

Calcula el valor signal XOR mask

(Comprueba que los bits de signal se conservan en las posiciones de valor 0 en mask, mientras que cambian de valor en las posiciones con valor 1 en mask)

[72]:
```
resultado = signal ^ mask

fv.print_int_bits(signal,formato='bin',num_bits=8)
fv.print_int_bits(mask,formato='bin',num_bits=8)
fv.print_int_bits(resultado,formato='bin',num_bits=8)
```

```
10010110b :: 150
00001111b :: 15
10011001b :: 153
```

Operaciones lógicas y de comparación

Resumen: Tabla de verdad de las operaciones lógicas:

A	B	A AND B	A OR B	NOT A
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False

- Las operaciones de comparación devuelven como resultado un valor booleano (***True***, ***False***)
- Empleo de **varios operadores de comparación consecutivos** para simplificar expresiones complejas:
  `(a comparado con b) and (b comparado con c)`
  es equivalente a
  `a comparado con b comparado con c`

```
# Ejemplo de expresiones equivalentes:
(3<4) and (4<5) and (5>=6)
3<4<5>=6
```

Inicializa las siguientes variables con los valores que se indican:

- temperatura: 25 grados
- umbral_temperatura: 28 grados
- dia_soleado: falso
- dia_lluvioso: falso

[73]:
```
temperatura=25
umbral_temperatura=28
dia_soleado=False
dia_lluvioso=False
```

Comprueba si la temperatura es mayor de 15 grados

[74]:
```
temperatura>15
```

[74]:
```
True
```

Comprueba si el dia es soleado

[75]:
```
dia_soleado == True
```

[75]:
```
False
```

Comprueba si la temperatura es menor de 30 grados y, además, el día es lluvioso

[76]:
```
# Alternativas posibles:
(temperatura<30) and (dia_lluvioso==True)

''' Nota: Los paréntesis no son necesarios porque las operaciones lógicas
tienen menor precedencia que los operaciones de comparación'''
temperatura<30 and dia_lluvioso==True
```

[76]:
```
False
```

Verifica que no hace sol

[77]:
```
# Alternativas posibles:
dia_soleado != True
dia_soleado == False
not (dia_soleado == True)
not (dia_soleado) # No es necesaria la comparación, pues dia_soleado es un valor␣
 ↪booleano
```

[77]:
```
True
```

Observa cómo afecta la **precedencia de las operaciones and y or** a los siguientes dos ejemplos:

[78]:
```
True or False and False
```

[78]: True

[79]:
```
(True or False) and False
```

[79]: False

Comprueba si la temperatura es menor que umbral_temperatura y, además, no hace sol

[80]:
```
# Alternativas:
(temperatura<umbral_temperatura) and (not dia_soleado)
temperatura<umbral_temperatura and not dia_soleado
```

[80]: True

Para decidir ir a la playa, es suficiente tener un **día soleado o bien que la temperatura sea mayor o igual que 25 grados**. Comprueba si se reunen las condiciones para ir a la playa

[81]:
```
dia_soleado or temperatura>=25
```

[81]: True

Para bañarse en la playa, se requiere, además de las condiciones para ir a la playa, que no esté lloviendo. Comprueba si se reunen las condiciones para bañarse en la playa

[82]:
```
(dia_soleado or temperatura>=25) and not dia_lluvioso
```

[82]: True

Para pasear al perro por la playa es necesario que se cumplan las dos condiciones siguientes: * No hay dia soleado o bien la temperatura es menor de 18 grados * El día es lluvioso

Comprueba si se puede llevar al perro a la playa

[83]:
```
(not dia_soleado or temperatura<18) and dia_lluvioso
```

[83]: False

Ejemplo de varios operadores de comparación consecutivos:

[84]:
```
3<4<5 # Equivale a (3<4) and (4<5)
```

[84]: True

[85]:
```
3<4 and 4<5
```

[85]: True

[86]:
```
3==2+1==3+0>=3.0 # Equivale  a (3==2+1) and (2+1==3+0) and (3+0>=3.0)
```

[86]: True

[87]:
```
3==2+1 and 2+1==3+0 and 3+0>=3.0
```

[87]: True

Asignación de variables con operador

Son equivalentes las siguientes dos formas de asignar valores a variables:

- `var operador = expresión`
- `var = var operador expresión`

[88]:
```
a=5
a+=2
a
```

[88]: 7

[89]:
```
a-=3
a
```

[89]: 4

[90]:
```
a*=a
a
```

[90]: 16

[91]:
```
a//=2
a
```

[91]: 8

4 Cadenas de caracteres (*strings*)

- Cadena de caracteres: secuencia de letras, números, espacios y/o signos de puntuación
- Las cadenas de caracteres van encerradas entre comillas (simples ['] o dobles ["])

[92]:
```
'Soy una cadena de caracteres'
```

[92]: 'Soy una cadena de caracteres'

[93]:
```
"Yo también soy un string"
```

[93]: 'Yo también soy un string'

4.1 Función *print*

- Muestra por pantalla una cadena de caracteres o varias separadas por un espacio blanco
- Finaliza con un salto de línea

  `print (string_1[,string_2,...,string_n])`

- Impresión de **caracteres especiales**:
  - \n: carácter *nueva línea*
  - \t: tabulador
  - \": Carácter "
  - \': Carácter '
    Nota: No es necesario *escapar* el carácter ' en cadenas delimitadas mediante ", ni el carácter (") cuando el delimitador es '

[94]:
```
str1="¿Qué es la Programación Orientada a Objetos?"
print(str1)
```

```
¿Qué es la Programación Orientada a Objetos?
```

[95]:
```
my_name='Guido van Rossum'
language="Python"

# Mostrar por pantalla un mensaje:
print(my_name, "es el creador del lenguaje",language)
```

```
Guido van Rossum es el creador del lenguaje Python
```

4.2 Concatenación de cadenas: +

[96]:
```
nombre="Augusta Ada King-Noel"
titulo="Condesa de Lovelace"
profesion='Matemática'

texto=nombre+' ('+profesion+', '+titulo+')'
print(texto,'es considerada como \nla primera programadora de ordenadores de la␣
 ↪historia')
```

```
Augusta Ada King-Noel (Matemática, Condesa de Lovelace) es considerada como
la primera programadora de ordenadores de la historia
```

4.3 Concatenación de cadenas repetidas: *

- `string * k` → Genera una cadena en la que `string` se repite `k` veces

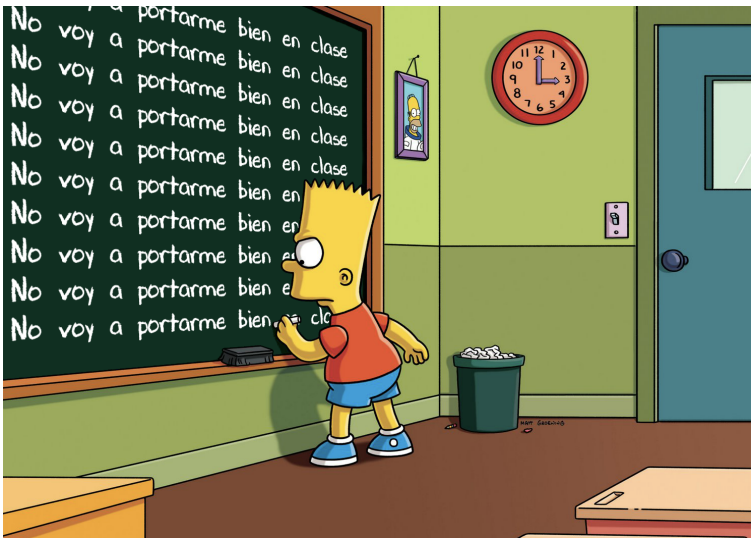

Imagen generada en: http://www.ranzey.com/generators/bart/index.html

[97]:
```
print('No voy a portarme bien en clase'*10)
```

```
No voy a portarme bien en claseNo voy a portarme bien en claseNo voy a portarme
bien en claseNo voy a portarme bien en claseNo voy a portarme bien en claseNo
voy a portarme bien en claseNo voy a portarme bien en claseNo voy a portarme
bien en claseNo voy a portarme bien en claseNo voy a portarme bien en clase
```

[98]:
```
print('No voy a portarme bien en clase\n'*10)  # \n : carácter nueva línea
```

```
No voy a portarme bien en clase
No voy a portarme bien en clase
No voy a portarme bien en clase
No voy a portarme bien en clase
No voy a portarme bien en clase
No voy a portarme bien en clase
No voy a portarme bien en clase
No voy a portarme bien en clase
No voy a portarme bien en clase
No voy a portarme bien en clase
```

[99]:
```
print("No trago al 'profe' de Mates")
```

```
No trago al 'profe' de Mates
```

[100]:
```
print("Me cae bien el "Topo", y explica muy bien") #Error: falta escapar las␣
 ↪comillas de "Topo"
```

```
  File "<ipython-input-100-3bc7301e66c8>", line 1
    print("Me cae bien el "Topo", y explica muy bien") #Error: falta escapar␣
↪las comillas de "Topo"
                                  ^
SyntaxError: invalid syntax
```

4.4 Comparación de cadenas

- Igualdad y desigualdad (== y !=)
  - **cadena_1 == cadena_2** → *True* si las dos cadenas son **iguales carácter a carácter**
  - **cadena_1 != cadena_2** → *True* si ambas cadenas **no son exactamente iguales**
- El resto de comparaciones (<, <=, >, >=) utilizan como criterio el **orden alfabético**
- El orden alfabético responde a los valores numéricos de la **tabla ASCII**:
  - Las letras mayúsculas tienen un código menor que las letras minúsculas
  - Las letras con tilde tienen un valor mayor que las letras sin tilde
  - Los números tienen valor menor que las letras

[101]:
```
'Europa'!='Oceanía'
```

[101]: True

[102]:
```
'América'<'Asia'
```

[102]: True

[103]:
```
"África"<'África'
```

[103]: True

[104]:
```
'José'>='Jose'
```

[104]: True

[105]:
```
'uno'>'2'
```

[105]: True

4.5 Funciones *ord* y *chr*

- `ord(carácter)` → devuelve el valor ASCII de un carácter
- `chr(valor_entero)` → devuelve el carácter correspondiente a un valor numérico

[106]:
```
ord('a')
```

[106]: 97

[107]:
```
ord('A')
```

[107]: 65

[108]:
```
chr(97)
```

[108]: 'a'

[109]:
```
chr(65)
```

[109]: 'A'

4.6 Métodos de las cadenas de caracteres

- Método: función propia de un objeto
- Forma de ejecutar un método: `dato.metodo(argumento1, argumento2,..., argumento n)`
- Algunos métodos propios de las cadenas de caracteres:
  - *lower*
  - *upper*
  - *title*
  - *replace*

[110]:
```
mi_cadena='--CADENA con MAYÚSCULAS y minúsculas--'; print(mi_cadena)
```

```
--CADENA con MAYÚSCULAS y minúsculas--
```

[111]:
```
mi_cadena1=mi_cadena.lower(); print(mi_cadena1)
```

```
--cadena con mayúsculas y minúsculas--
```

[112]:
```
mi_cadena3=mi_cadena.upper(); print(mi_cadena3)
```

```
--CADENA CON MAYÚSCULAS Y MINÚSCULAS--
```

[113]:
```
mi_cadena4=mi_cadena.title(); print(mi_cadena4)
```

```
--Cadena Con Mayúsculas Y Minúsculas--
```

[114]:
```
mi_cadena5=mi_cadena.replace('--','==='); print(mi_cadena5)
```

```
===CADENA con MAYÚSCULAS y minúsculas===
```

5 Funciones predefinidas

```
nombre_de_función (argumentos_separados_por_comas)
```

```
abs(valor_numérico) # calcula el valor absoluto
float(valor_numérico) # convierte a float
float(cadena_caracteres_numérica) # convierte a float
int(valor_numérico) # convierte a entero
int(cadena_caracteres_numérica) # convierte a entero
str(valor_numérico) # convierte a cadena de caracteres
round(valor_numérico) # redondea el valor numérico
round(valor_numérico, numero_de_decimales) # redondea con los decimales indicados
```

Las funciones pueden combinarse con otros operadores y operaciones

[115]:
```
abs(-5.4)
```

[115]:
```
5.4
```

[116]:
```
float(3)
```

[116]:
```
3.0
```

[117]:
```
float('6.32')
```

[117]:
```
6.32
```

[118]:
```
int(5.23)
```

[118]:
```
5
```

[119]:
```
int('38')
```

[119]:
```
38
```

[120]:
```
str(12)
```

[120]:
```
'12'
```

[121]:
```
round(7.435123)
```

[121]:
```
7
```

[122]:
```
round(7.435123,2)
```

[122]:
```
7.44
```

[123]:
```
float('1.5'+'63')
```

[123]: 1.563

```
[124]: str(int(1.5)+float(6.1))
```

[124]: '7.1'

6 Funciones de módulos

```
from nombre_del_módulo import nombre_de_función
from nombre_del_módulo import nombres_de_funciones_separados_por_comas
from nombre_del_módulo import *
```

6.1 Librería matemática: *math*

```
[125]: from math import sin, cos, tan, exp, ceil, floor, log, log10, sqrt
       from math import pi, e
```

```
[126]: a=sin(3); print(a)
```

0.1411200080598672

```
[127]: b=cos(pi); print(b)
```

-1.0

```
[128]: c=tan(1.2); print(c)
```

2.5721516221263183

```
[129]: d=exp(1); print(d)
```

2.718281828459045

```
[130]: f=ceil(2.1); print(f)
```

3

```
[131]: g=floor(1.2); print(g)
```

1

```
[132]: h=log(d); print(h)
```

1.0

[133]:
```
i=log10(1); print(i)
```

```
0.0
```

[134]:
```
j=sqrt(64); print(j)
```

```
8.0
```