Lenguaje de Programación Python

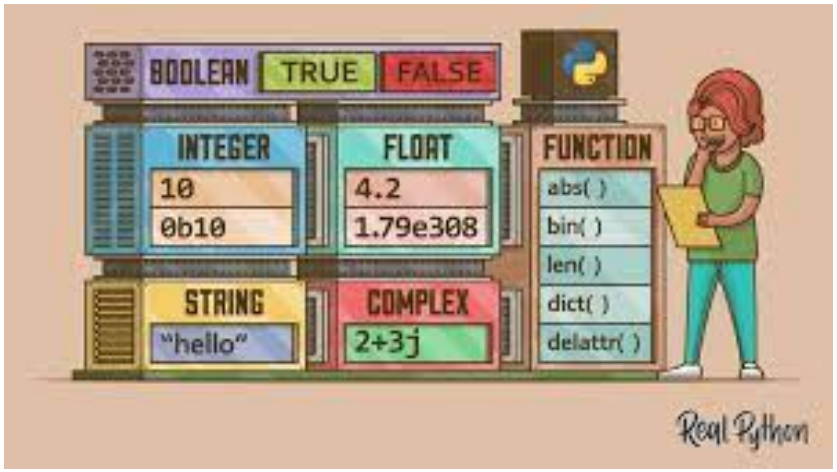
Paradigma Imperativo-Tipos de Datos Básicos

Dr. Mario Marcelo Berón

Universidad Nacional de San Luis
Departamento de Informática
Área de Programación y Metodologías de Desarrollo de Software

1. Enteros
2. Booleanos
3. Flotantes
4. Complejos
5. Decimal

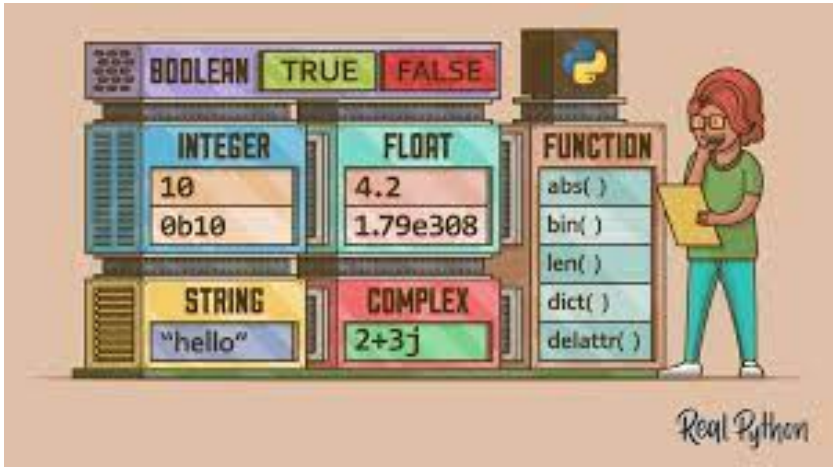
Python: Tipos Integrales



Los *Tipos Integrales* que proporciona Python son: *int* y *bool*.

- Ambos son inmutables.
- Para el caso del tipo bool 0 representa falso y otro valor distinto de cero se interpreta como verdadero. Como se verá más adelante también existen *True* y *False*.

Python: Tipos Integrales - Enteros



- El tamaño de un entero está limitado por el tamaño de la memoria de la máquina.
- Se pueden crear enteros usando el constructor *int()* o bien por medio de literales.

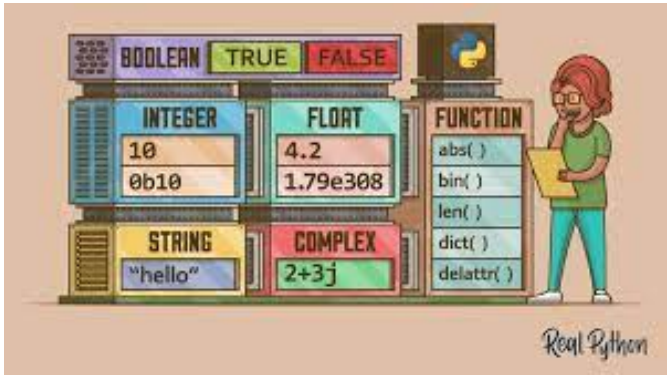
Python: Tipos Integrales - Enteros

Syntax	Description
`x + y`	Adds number `x` and number `y`
`x - y`	Subtracts `y` from `x`
`x * y`	Multiplies `x` by `y`
`x / y`	Divides `x` by `y`; always produces a `float` (or a `complex` if `x` or `y` is `complex`)
`x // y`	Divides `x` by `y`; truncates any fractional part so always produces an `int` result; see also the `round()` function
`x % y`	Produces the modulus (remainder) of dividing `x` by `y`
`x ** y`	Raises `x` to the power of `y`; see also the `pow()` functions
`-x`	Negates `x`; changes `x`'s sign if nonzero, does nothing if zero
`+x`	Does nothing; is sometimes used to clarify code
`abs(x)`	Returns the absolute value of `x`
`divmod(x, y)`	Returns the quotient and remainder of dividing `x` by `y` as a tuple of two `ints`
`pow(x, y)`	Raises `x` to the power of `y`; the same as the `**` operator
`pow(x, y, z)`	A faster alternative to `(x ** y) % z`
`round(x, n)`	Returns `x` rounded to *n* integral digits if *n* is a negative `int` or returns `x` rounded to *n* decimal places if *n* is a positive `int`; the returned value has the same type as `x`; see the text

Python: Tipos Integrales - Enteros

Syntax	Description
`bin(i)`	Returns the binary representation of `int i` as a string, e.g., `bin(1980) == '0b11110111100'`
`hex(i)`	Returns the hexadecimal representation of `i` as a string, e.g., `hex(1980) == '0x7bc'`
`int(x)`	Converts object `x` to an integer; raises `ValueError` on failure—or `TypeError` if `x`'s data type does not support integer conversion. If `x` is a floating-point number it is truncated.
`int(s, base)`	Converts `str s` to an integer; raises `ValueError` on failure. If the optional *base* argument is given it should be an integer between 2 and 36 inclusive.
`oct(i)`	Returns the octal representation of `i` as a string, e.g., `oct(1980) == '0o3674'`

Python: Tipos Integrales - bool



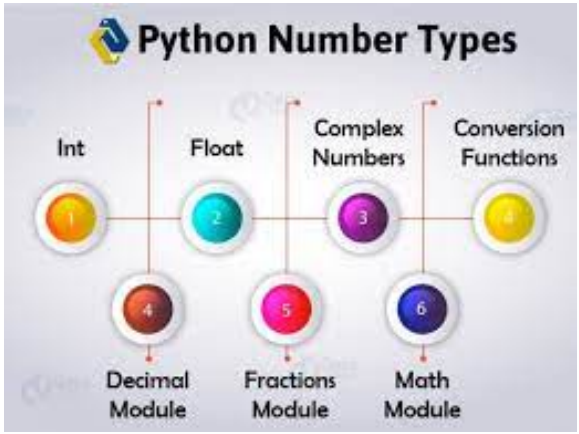
- Existen dos objetos booleanos: *True* y *False*.
- El tipo tiene un constructor denominado *bool()* el cual cuando se invoca sin argumentos retorna como resultado *False*. Con un argumento de tipo *bool* retorna una copia del argumento.
- Todos los tipos primitivos se pueden convertir a boolean y es fácil proveer conversiones booleanas para tipos personalizados.
- Python provee tres operadores lógicos: *and*, *or* y *not*. Los dos primeros utilizan la lógica de corto circuito y retornan el operando que determinó el resultado.

Python: Tipos Integrales - bool

Ejemplo

```
>>> t= True
>>> f= False
>>> t and f
False
>>> t and True
True
```

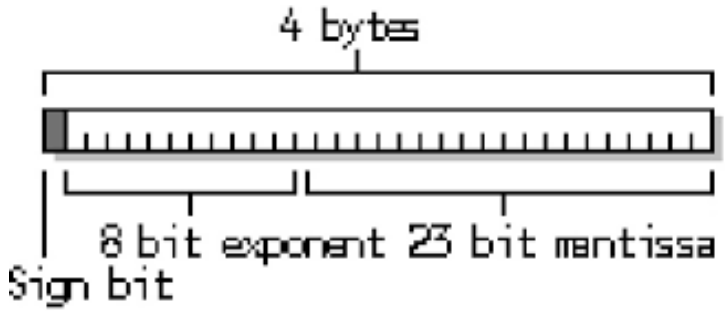
Python: Punto Flotante



Python provee tres clases de valores de punto flotante, a saber:

- *float* almacena números de punto flotante doble precisión cuyo rango depende del compilador con el que se construyó Python. Se escriben usando un punto decimal o notación exponencial.
- *decimal* se utiliza cuando se requiere una alta precisión.
- *complex*.

Python: Punto Flotante-float



Punto Flotante: *float*

- Mantiene un número en punto flotante de doble precisión.
- Tiene un constructor denominado float el cual cuando se invoca sin argumentos crea un número en punto flotante inicializado en cero. Cuando se invoca con un argumento retorna la copia de dicho argumento.
- Cuando el constructor se usa para conversiones puede recibir un argumento string con notación decimal o exponencial.
- Las operaciones antes vistas para enteros presentadas con anterioridad se pueden usar para los floats.

Python: Punto Flotante-Complex

Complex:

- Es un tipo de dato inmutable que mantiene dos números flotantes que representan la parte real y la imaginaria.
- Los literales complejos son escritos con la parte real y la parte imaginaria unida por los signos + o -, y con la parte imaginaria seguida por una j.
- Las partes real e imaginaria de un complejo están disponibles a través de los atributos real e imag.

Python: Punto Flotante-Complex

Complex:

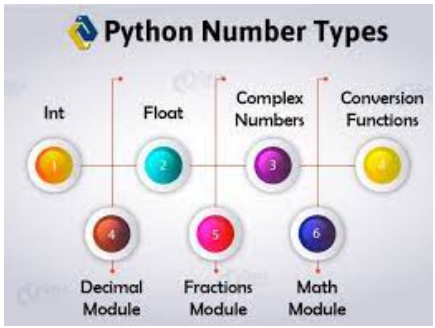
- Excepto por //, %, *divmod()* y *pow()* esta última con tres argumentos, todas los operadores numéricos y funciones provistos por Python para números se pueden utilizar.
- Tiene un constructor denominado *complex* el cual cuando se invoca sin argumentos retorna como resultado 0j. Cuando recibe un complejo retorna una copia del argumento y con cualquier otro argumento intenta convertirlo a complejo.

Python: Punto Flotante-Complex

Ejemplo

```
>>> c1=complex(1,2)
>>> c1
(1+2j)
>>> c2=complex(2,3)
>>> c1+c2
(3+5j)
>>>
```

Decimal:



- Proveen la precisión deseada por el programador.
- Son inmutables.
- Los cálculos que usan decimales son más lentos que los realizados con flotantes.
- Tienen un constructor denominado *decimal()*. Esta función puede tomar un argumento entero o un string pero no un flotante (por cuestiones de precisión).
- Todas las operaciones descritas para números se pueden utilizar para los decimales.
- Son más lentos que los *float*.

Punto Flotante: Decimal

```
>>> import decimal
>>> a = decimal.Decimal(9876)
>>> b = decimal.Decimal("54321.01234567898765432 1")
>>> a + b
Decimal('64197.012345678987654321')
```

Decimal

Algunas veces la diferencia de precisión se nota.

```
>>> 23 / 1.05
21.904761904761905
>>> print(23 / 1.05)
21.9047619048
>>> print(decimal.Decimal(23) / decimal.Decimal("1.05"
21.904761904761904761904761905
>>> decimal.Decimal(23) / decimal.Decimal("1.05")
Decimal('21.904761904761904761904761905')
```