Python Datatypes

Thursday, 8 December 2022

Today's Agenda

- Basic Data Types In Python

- Numeric Types

- Different Types Of Integers

- The **float** Type

- The **complex** Type

- The **bool** Type

- The **str** Type

Basic Data Types In Python

- Although a **programmer is not allowed to mention the data type** while creating variables in his program in **Python** , but **Python** internally allots different data types to variables depending on their declaration style and values.

- Overall **Python** has **14 data types** and these are classified into **6 categories.**

Basic Data Types In Python

- These categories are:

  - **Numeric Types**
  - **Boolean Type**
  - **Sequence Types**
  - **Set Types**
  - **Mapping Type**
  - **None Type**

- Given on the next slide are the names of actual data types belonging to the above mentioned categories

Basic Data Types In Python

Numeric Type	Boolean Type	Sequence Type	Set Type	Mapping Type	None Type
int	bool	str	set	dict	NoneType
float		list	frozenset		
complex		bytes			
		bytearray			
		tuple			
		range			

Some Very Important Points

- Before we explore more about these data types , let us understand following important points regarding Python's data types:

1. **DATA TYPES IN PYTHON ARE DYNAMIC**

1. **SIZE OF THE DATA TYPE IS ALSO DYNAMICALLY MANAGED**

1. **DATA TYPES ARE UNBOUNDED**

Some Very Important Points

1. **DATA TYPES IN PYTHON ARE DYNAMIC**

- The term **dynamic** means that we can assign different values to the same variable at different points of time.

- Python will dynamically change the type of variable as per the value given.

Some Very Important Points

Some Very Important Points

2. **SIZE OF THE DATA TYPE IS ALSO DYNAMICALLY MANAGED**

- In **Python** the size of **data types** is **dynamically managed**

- Like **C/C++/Java** language , variables in **Python** are **not of fixed size**.

- **Python makes them as big as required** on demand

- There is no question of how much memory a variable uses in **Python** because **this memory increases as per the value being assigned**

Some Very Important Points

- **Python** starts with **initial size** for a variable and then increases its size as needed up to the **RAM limit**

- This initial size for **int** is **24 bytes** and then increases as the value is increased

- If we want to check the size of a variable , then **Python** provides us a function called **getsizeof()** **.**

- This function is available in a module called **sys**

Some Very Important Points


```
>>> import sys
>>> sys.getsizeof(0)
24
>>> sys.getsizeof(1)
28
>>> sys.getsizeof(123456789123456789123456789123456789)
40
>>>
```

Some Very Important Points

3. **DATA TYPES ARE UNBOUNDED**

- Third important rule to remember is that , in **Python** data types like **integers** don't have any range i.e. **they are unbounded**

- **Like C /C++ /Java they don't have max or min value**

- So an **int** variable can store **as many digits as we want**.

Numeric Types In Python

As previously mentioned , Python supports **3 numeric types:**

int: Used for storing integer numbers without any fractional part

float: Used for storing fractional numbers

complex: Used for storing complex numbers

Numeric Types In Python

EXAMPLES OF int TYPE:

a=10

b=256

c=-4

print(a)

print(b)

print(c)

The float Data Type

- **Python** also supports **floating-point real values**.

- Float values are specified with a **decimal point**

- So **2.5** , **3.14** , **6.9** etc are all examples of **float** data type

- Just like double data type of other languages like **Java/C** , float in **Python** has a precision of **16 digits**

Some Important Points About float

- Float values can also be represented as **exponential** values

- Exponential notation is a scientific notation which is represented using **e** or **E** followed by an integer and it means to the **power of 10**

```
>>> a=3.5e4
>>> a
35000.0
```

The complex Data Type

- Complex numbers are written in the form, **x + yj**, where **x** is the **real part** and **y** is the **imaginary part**.

- For example: **4+3j** , **12+1j** etc

- The letter **j** is called **unit imaginary number.**

- It denotes the value of **√-1** , i.e **j²** denotes **-1**

An Example

```
>>> a=2+3j
>>> print(a)
(2+3j)
>>> type(a)
<class 'complex'>
```

Some Important Points About complex Data Type

- For representing the **unit imaginary number** we are only allowed to use the letter **j (both upper and lower case are allowed)**.

- Any other letter if used will generate error

```
>>> a=2+3i
  File "<stdin>", line 1
    a=2+3i
         ^
SyntaxError: invalid syntax
```

Some Important Points About complex Data Type

The letter **j** , should only appear in suffix , not in prefix

```
>>> a=2+j3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'j3' is not defined
```

Some Important Points About complex Data Type

The **real** and **imaginary** parts are allowed to be **integers** as well as **floats**

```
>>> a=1.5+2.6j
>>> print(a)
(1.5+2.6j)
```

Some Important Points About complex Data Type

- We can display **real** and **imaginary** part separately by using the attributes of complex types called **"real"** and **"imag".**

```
>>> a=2+5j
>>> print(a.real)
2.0
>>> print(a.imag)
5.0
```

- Don't think **real** and **imag** are functions , rather they are **attributes/properties** of **complex data type**

The bool Data Type

- In Python , to represent **Boolean** values we have **bool data type.**

- The **bool data type** can be one of two values, either **True** or **False**.

- We use Booleans in programming to make comparisons and to control the flow of the program.

Some Examples

```
>>> a=False
>>> print(a)
False
```

```
>>> a=False
>>> type(a)
<class 'bool'>
```

Some Important Points About bool

True and False are keywords , so case sensitivity must be remembered while assigning them otherwise Python will give error

```
>>> a=false
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'false' is not defined
```

Some Important Points About bool

○ All test conditions in **Python** return the result as **bool** which could be either **True** or **False**

```
>>> a=10
>>> b=5
>>> print(a>b)
True
```

```
>>> x=15
>>> y=15
>>> print(x<y)
False
```

Some Important Points About bool

To understand the next point , try to guess the output of the following:

```
a=True
b=True
c=a+b
print(c)
```

```
a=False
b=False
c=a+b
print(c)
```

```
a=True
b=False
c=a+b
print(c)
```

The above outputs make it clear that internally **Python** stores **True** and **False** as **integers** with the value **1** and **0** respectively

The str Data Type

- Just like any other language , In **Python** also a **String** is sequence of characters.

- **Python** does not have a **char data type**, unlike **C/C++** or **Java**

- We can use **single quotes** or **double quotes** to represent strings.

- However **Python** recommends to use **single quotes**

Some Examples


```
>>> name='Sachin'
>>> print(name)
Sachin
```

```
>>> name="Sachin"
>>> type(name)
<class 'str'>
```

```
>>> name="Sachin"
>>> print(name)
Sachin
```

The data type used by **Python** internally for storing Strings is **str**

Some Important Points About Strings

- Unlike **C language** , **Python** does not uses **ASCII** number system for characters . It uses **UNICODE** number system

- **UNICODE** is a number system which supports much wider range of characters compared to **ASCII**

- As far as Python is concerned , it uses **UNICODE** to support **65536** characters with their numeric values ranging from **0** to **65535** which covers almost every spoken language in the world like **English** , **Greek** , **Spanish** , **Chinese** , **Japanese** etc

Some Important Points About Strings

To quote the unicode website they are atleast 61 different languages supported.
http://www.lexilogos.com/keyboard/index.htm

```
Какво е Unicode ? in Bulgarian (30 letters)
Što je Unicode? in Croatian (30 letters)
Co je Unicode? in Czech (48 letters)
Hvad er Unicode? in Danish(29 letters)
Wat is Unicode? in Dutch(26 letters)
□□□□ □□ □□□□□□□? in English (Deseret)
□□□ □□ □□□□□□? in English (Shavian)
Kio estas Unikodo? in Esperanto(31 letters)
Mikä on Unicode? in Finnish(29 letters)
Qu'est ce qu'Unicode? in French
რა არის უნიკოდი? in Georgian
Was ist Unicode? in German
Τι είναι το Unicode; in Greek (Monotonic)
Τί εῖναι τὸ Unicode; in Greek (Polytonic)
מהו יוניקוד (Unicode)? in Hebrew
यूनिकोड क्या है? in Hindi
Mi az Unicode? in Hungarian
Hvað er Unicode? in Icelandic
Gịnị bụ Yunikod? in Igbo
Que es Unicode? in Interlingua
Cos'è Unicode? in Italian
ユニコードとはか？in Japanese
ಯುನಿಕೋಡ್ ಎಂದರೇನು? in Kannada
유니코드에 대해? in Korean
Kas tai yra Unikodas? in Lithuanian
Што е Unicode? in Macedonian
X'inhu l-Unicode? in Maltese
Unicode гэж юу вэ? in Mongolian
युनिकोड के हो? in Nepali
Unicode, qu'es aquò? in Occitan
یونیکد چیست؟ in Persian
Czym jest Unikod? in Polish
O que é Unicode? in Portuguese
```

Some Important Points About Strings

- If a string starts with **double quotes** , it must end with **double quotes** only .

- Similarly if it starts with **single quotes** , it must end with **single quotes** only.

- Otherwise **Python** will generate **error**

Some Important Points About Strings

```
>>> s="Welcome"
>>> print(s)
Welcome
>>> s="Welcome'
  File "<stdin>", line 1
    s="Welcome'
              ^
SyntaxError: EOL while scanning string literal
```

Some Important Points About Strings

- If the string contains **single quotes** in between then it must be enclosed in **double quotes** and **vice versa**.

- **For example:**

- To print **Sunny's Python Classes** , we would write:

  msg= " Sunny's Python Classes "

- Similarly to print **Capital of "MP" is "Bhopal"** ,we would write:

  msg= 'Capital of "MP" is "Bhopal" '

Some Important Points
About Strings

```
>>> msg="Sachin's Python Classes"
>>> print(msg)
Sachin's Python Classes
```

```
>>> msg='Capital of "MP" is "Bhopal"'
>>> print(msg)
Capital of "MP" is "Bhopal"
```

Some Important Points About Strings

- How will you print **Let's learn "Python"** ?

A. **"Let's learn "Python" "**

A. **'Let's learn "Python" '**

NONE!
Both will give error.
Correct way is to use either **triple single quotes** or **triple double quotes** or **escape sequence character **
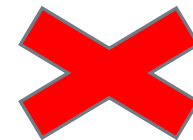**msg=' ' 'Let's learn "Python" ' ' '**
OR
msg='Let\'s learn "Python" '

Some Important Points About Strings

```
>>> msg='Let's learn "Python" '
  File "<stdin>", line 1
    msg='Let's learn "Python" '
              ^
SyntaxError: invalid syntax
```



```
>>> msg="Let's learn "Python""
  File "<stdin>", line 1
    msg="Let's learn "Python""
                          ^
SyntaxError: invalid syntax
```
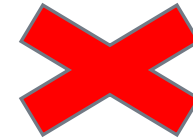
Some Important Points About Strings

```
>>> msg='''Let's learn "Python"'''
>>> print(msg)
Let's learn "Python"
```



```
>>> msg='Let\'s learn "Python" '
>>> print(msg)
Let's learn "Python"
```

Some Important Points About Strings

Another important use of **triple single quotes** or **triple double quotes** is that if our string extends up to more than one line then we need to enclose it in **triple single quotes** or **triple double quotes**

**A = """my
name
is
sunny"""**

A = "my\n name\n is\n sunny"

Accessing Individual Characters In String

In **Python**, Strings are stored as individual characters in a contiguous memory location.

Each character in this memory location is assigned an index which begins from **0** and goes up to **length -1**

Accessing Individual Characters In String

For example, suppose we write

word="Python"

Then the internal representation of this will be

	0	1	2	3	4	5
word	P	Y	T	H	O	N

Accessing Individual Characters In String

Now to access individual character we can provide this **index number** to the **subscript operator [ ]**.

```
>>> word="Python"
>>> print(word[0])
P
>>> print(word[1])
y
>>> print(word[2])
t
```

Accessing Individual Characters In String

However if we try to provide an index number beyond the given limit then **IndexError** exception will arise

```
>>> word="Python"
>>> print(word[7])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
>>>
```

Accessing Individual Characters In String

Not only this , Python even allows negative indexing which begins from the end of the string.

So **-1** is the index of last character , **-2** is the index of second last character and so on.

Forward Indexing

	0	1	2	3	4	5

word	P	Y	T	H	O	N
	-6	-5	-4	-3	-2	-1

Backward Indexing
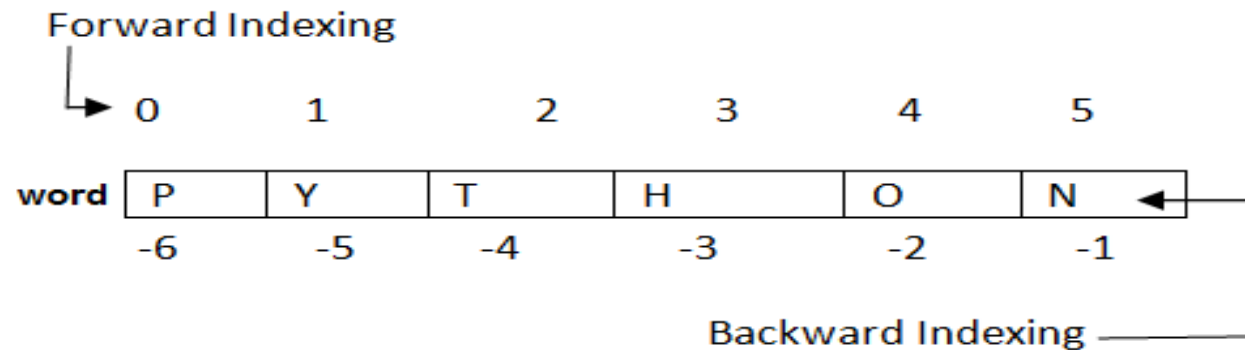
Accessing Individual Characters In String

```
>>> word="Python"
>>> print(word[-1])
n
>>> print(word[-2])
o
```

Type Conversion

The process of converting the value of one data type (integer, string, float, etc.) to another data type is called **Type Conversion**.

Python has **two** types of **type conversion**.

◦ **Implicit Type Conversion**

◦ **Explicit Type Conversion**

Implicit Conversion

In **Implicit Type Conversion**, **Python** automatically converts one data type to another data type.

This process doesn't need any programmer involvement.

Let's see an example where **Python** promotes conversion of **int** to **float** .

Example Of Implicit Conversion


```
>>>   a=10
>>>   b=6.5
>>>   c=a+b
>>>   print(a)
10
>>>   print(b)
6.5
>>>   print(c)
16.5
>>>   print(type(c))
<class 'float'>
```

- If we observe the above operations , we will find that **Python** has automatically assigned the data type of **c** to be **float** .

- This is because **Python** always converts **smaller data type** to **larger data type** to avoid the loss of data.

Another Example

```
>>> a=10
>>> b=True
>>> c=a+b
>>> print(a)
10
>>> print(b)
True
>>> print(c)
11
>>> print(type(c))
<class 'int'>
```

- Here also **Python** is automatically upgrading **bool** to type **int** so as to make the result sensible

Explicit Type Conversion

- There are some cases , where **Python** will not perform type conversion automatically and we will have to explicitly convert one type to another.

- Such **Type Conversions** are called **Explicit Type Conversion**

- Let's see an example of this

Explicit Type Conversion

Guess the output ?

```
a=10
b="6"
print(type(a))
print(type(b))
c=a+b
print( c )
print(type( c ))
```

Output:
<class 'int'>
<class 'str'>
TypeError: unsupported operand type(s) for +: 'int' and 'str'

Why did the code fail?

The code **failed** because
Python does not automatically convert
String to **int**.

To handle such cases we need to
perform **Explicit Type Conversion**

Explicit Type Conversion Functions In Python

- Python provides us **5 predefined functions** for performing **Explicit Type Conversion** for fundamental data types.

- These functions are :
1. **int()**
2. **float()**
3. **complex()**
4. **bool()**
5. **str()**

The **int( )** Function

- **Syntax:** **int(value)**

- This function converts **value of any data type to integer ,** *with some special cases*

- It returns an **integer** object converted from the given **value**
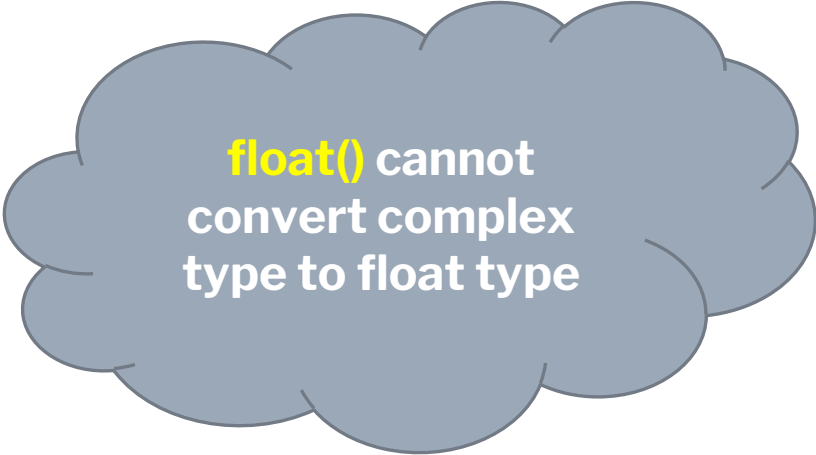
int( ) Examples

int(2.3)
Output:
2
int(False)
Output:
0
int(True)
Output:
1
int(3+4j)
Output:
TypeError: Can't convert complex to int
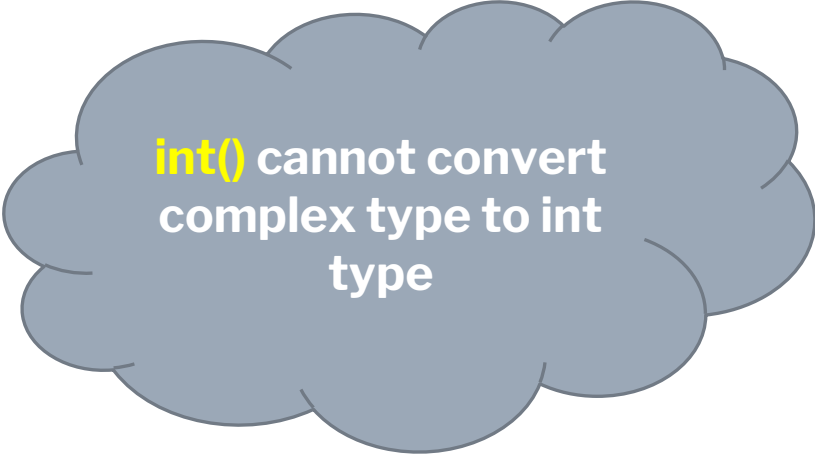
int() cannot convert complex type to int type

int( ) Examples

int("25")
Output:
25
int("2.5")
Output:
ValueError: Invalid literal for int()
int("1010")
Output:
1010
int("0b1010")
Output:
ValueError: Invalid literal for int()

int() cannot accept anything other than digits in a string

int() cannot accept binary values as string

Solution To The Previous Problem

Can you solve this error now ?

a=10

b="6"

c=a+b

print(c)

Output:

TypeError

Solution:

a=10

b="6"

c=a+int(b)

print(c)

Output:

16

The **float( )** Function

- **Syntax:** **float(value)**

- This function converts **value of any data type to float ,** *with some special cases*

- It returns an **float** object converted from the given **value**

float( ) Examples

float(25)
Output:
25.0
float(False)
Output:
0.0
float(True)
Output:
1.0
float(3+4j)
Output:
TypeError: Can't convert complex to float

float() cannot convert complex type to float

iNeuron

float( ) Examples

float("25")
Output:
25.0
float("2.5")
Output:
2.5
float("1010")
Output:
1010.0
float ("0b1010")
Output:
ValueError:Could not convert string to float
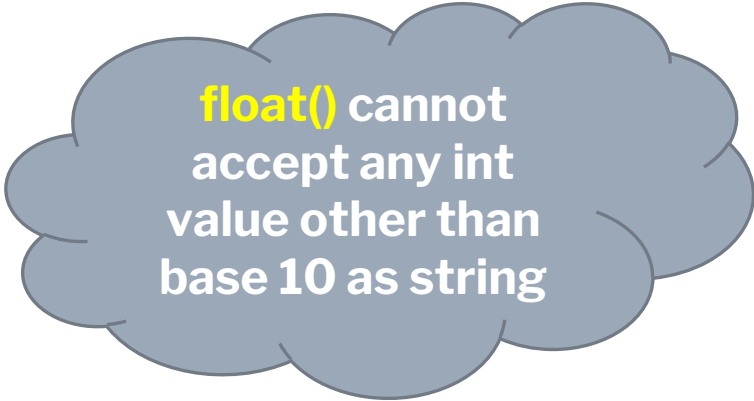
float("twenty")
Output:
ValueError:Could not convert string to float

float() cannot accept any int value other than base 10 as string

The **complex( )** Function

- **Syntax:** **complex(value)**

- This function converts **value of any data type to complex ,** *with some special cases*

- It returns an **complex** object converted from the given **value**

complex ( ) Examples

complex(25)
Output:
(25+0j)
complex(2.5)
Output:
(2.5+0j)
complex(True)
Output:
(1+0j)
complex(False)
Output:
0j

complex( ) Examples

complex("25")
Output:
(25+0j)
complex("2.5")
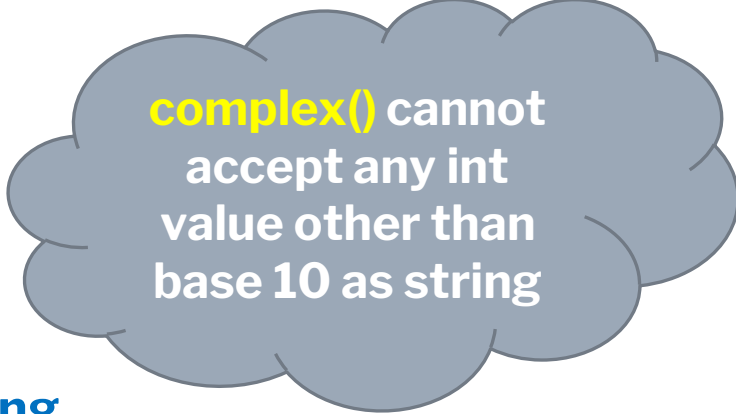Output:
(2.5+0j)
complex("1010")
Output:
(1010+0j)
complex ("0b1010")
Output:
ValueError: complex() arg is a malformed string

complex("twenty")
Output:
ValueError: complex() arg is a malformed string

complex() cannot accept any int value other than base 10 as string

The **bool ( )** Function

- **Syntax:** **bool(value)**

- This function converts **value of any data type to bool ,** *using the standard truth testing procedure.*

- It returns an **bool** object converted from the given **value**

The **bool ( )** Function

- **What values are considered to be false and what values are true ?**

- The following values are considered **false** in **Python**:
  - **None**
  - **False**
  - **Zero of any numeric type. For example, 0, 0.0, 0+0j**
  - **Empty sequence. For example: (), [], ''.**
  - **Empty mapping. For example: {}**

- **All other values are true**

bool( ) Examples

bool(1)
Output:
True
bool(5)
Output:
True
bool(0)
Output:
False
bool(0.0)
Output:
False

bool( ) Examples

bool(0.1)
Output:
True
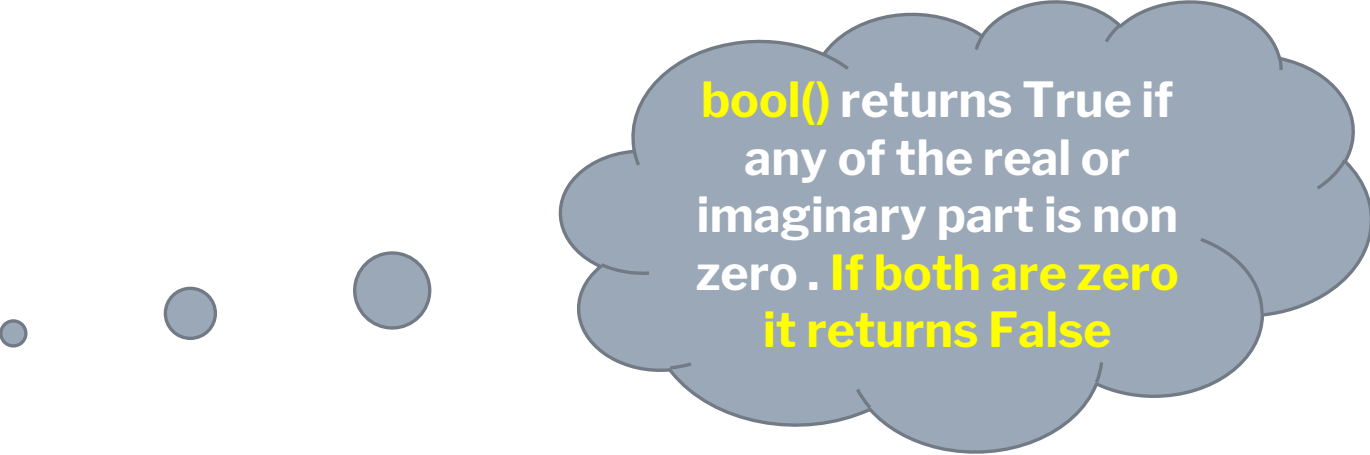bool(0b101)
Output:
True
bool(0b0000)
Output:
False
bool(2+3j)
Output:
True

bool() returns True if any of the real or imaginary part is non zero . If both are zero it returns False

bool( ) Examples

bool(0+1j)
Output:
True
bool(0+0j)
Output:
False
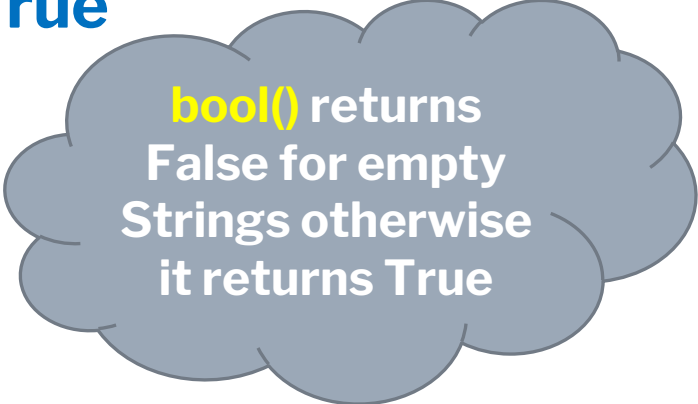bool('')
Output:
False
bool('A')
Output:
True

bool("twenty")
Output:
True
bool(' ')
Output:
True

bool() returns False for empty Strings otherwise it returns True

The **str( )** Function

- **Syntax:** **str(value)**

- This function converts **any data type to string ,** *without any special cases*

- It returns a **String** object converted from the given **value**

str( ) Examples

str(15)
Output:
'15'
str(2.5)
Output:
'2.5'
str(2+3j)
Output:
'(2+3j)'
str(True)
Output:
'True'

str( ) Examples

str(1)
Output:
'1'
str(5)
Output:
'5'
str(2.5)
Output:
'2.5'
str(True)
Output:
'True'

Thank you