Definition

Number variables are a fundamental concept in programming, and they're essential for any Python programmer to understand. In Python, there are two types of number variables: integers and floats. Integers are whole numbers, while floats are decimal numbers.

Types of Number Variables

In Python, there are several data types used to store numbers, including: **integers**, **floats** and **complex numbers.**

int (integer)

An integer is a whole number that doesn't have a decimal point. For example, 1, 2, 3, and 4 are all integers. In Python, you can declare an integer by simply assigning a whole number to a variable. For example:

In [1]:
```
my_int = 10
```

In [2]:
```
print('my_int :', my_int)

print(type(my_int))
```

```
my_int : 10
<class 'int'>
```

float (floating-point number)

A float, on the other hand, is a number that has a decimal point. For example, 1.5, 2.75, and 3.14159 are all floats. In Python, you can declare a float by including a decimal point in the number or by using scientific notation. For example:

In [3]:
```
my_float_1 = 3.14
my_float_2 = 2e-3  # this is equivalent to 0.002
```

In [4]:
```
print('my_float_2 :', my_float_2)

print(type(my_float_2))
```

```
my_float_2 : 0.002
<class 'float'>
```

complex (complex number)

These are used to store complex numbers, such as 2 + 3j, where j represents the imaginary unit. In Python, You can declare a complex number by including a imaginiry unit **j**. For example:

In [5]:
```
my_complex = 4.5 + 7j
```

In [6]:
```
print('my_complex :', my_complex)

print(type(my_complex))
```

```
my_complex : (4.5+7j)
<class 'complex'>
```

Note: It's important to note that Python automatically assigns the type of variable based on the value you assign to it. For example, if you assign a whole number to a variable, it will be an integer type. If you assign a number with a decimal point to a variable, it will be a float type.

Let's look at an example where we declare both an integer and a float variable:

In [7]:
```
my_int = 10
my_float = 3.14
```

In [8]:
```
print(type(my_int))
print(type(my_float))
```

```
<class 'int'>
<class 'float'>
```

Now that we have an understanding of three types of number variables in Python, we can move on to declaring and assigning variables.

Declaring and Assigning Variables

In Python, you can declare and assign number variables in a single line of code. To declare a variable, simply choose a name for it and use the equals sign (=) to assign a value to it.

Here's an example of how to declare and assign an integer variable:

In [9]:
```
my_int = 100
```

In the above example, we declared a variable called my_int and assigned it the value of 100. We can also declare and assign a float variable in the same way:

In [10]:
```
my_float = 7.56
```

In this example, we declared a variable called my_float and assigned it the value of 7.56

Note: It's important to note that variable names are case-sensitive in Python. This means that *my_int and My_int* are considered two different variables. Variable names can consist of letters, numbers, and underscores, but they cannot begin with a number.

We can also assign a new value to an existing variable simply by using the equals sign (=) again. For example:

In [11]:
```
my_int = 20
```

In [12]:
```
print(my_int)
```

```
20
```

In this example, we reassigned the value of the my_int variable to 20.

Now that we know how to declare and assign number variables in Python, let's move on to performing arithmetic operations with them.

Arithmetic Operations

Python provides basic arithmetic operators that you can use to perform arithmetic operations with number variables. Here are the basic arithmetic operators in Python:

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo (remainder)
**	Exponentiation

Let's see how we can use these operators with number variables in Python.

In [13]:
```
# Addition
x = 5
y = 3
result = x + y
print(result)  # Output: 8
```

```
8
```

In [14]:
```
# Subtraction
x = 5
y = 3
result = x - y
print(result)  # Output: 2
```

```
2
```

In [15]:
```
# Multiplication
x = 5
y = 3
result = x * y
print(result)  # Output: 15
```

```
15
```

In [16]:
```
# Division
x = 5
y = 3
result = x / y
print(result)  # Output: 1.6666666666666667
```

```
1.6666666666666667
```

In [17]:
```
# Modulo (remainder)
x = 5
y = 3
result = x % y
print(result)  # Output: 2
```

```
2
```

In [18]:
```
# Exponentiation
x = 5
y = 3
result = x ** y
print(result)  # Output: 125
```

```
125
```

In the above examples, we assigned values to two variables, x and y, and used the arithmetic operators to perform various arithmetic operations. We assigned the result of each operation to a new variable called result, and then printed out the value of result using the print() function.

Note: It's important to note that when performing division with integers, Python will return a float result. If you want to perform integer division, you can use the double slash (//) operator.

In [19]:
```
x = 5
y = 3
result = x // y
print(result)  # Output: 1
```

```
1
```

Now that we know how to perform arithmetic operations with number variables in Python, let's move on to comparing number variables.

Comparison Operators

Python provides comparison operators that you can use to compare number variables. Here are the comparison operators in Python:

Operator	Description
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Let's see how we can use these operators with number variables in Python.

In [20]:
```
x = 5
y = 3
print(x == y)  # Output: False
print(x != y)  # Output: True
print(x > y)   # Output: True
print(x < y)   # Output: False
print(x >= y)  # Output: True
print(x <= y)  # Output: False
```

```
False
True
True
False
True
False
```

In this example, we used the comparison operators to compare the values of x and y. Each comparison operator returns a boolean value (True or False) depending on whether the comparison is true or false.

Note: It's important to note that when comparing float variables, you may encounter issues with precision. This is because floats are stored as approximations in Python. To compare float variables, you may need to use a tolerance or a library that handles floating-point arithmetic.

Now that we know how to compare number variables in Python, let's move on to type conversion of python variables.

Type Conversion

Sometimes we may need to convert a number variable from one type to another. Python provides built-in functions that allow us to convert number variables from one type to another.

Here are the common built-in functions for type conversion in Python:

Function	Description
int(x)	Converts *x* to an integer
float(x)	Converts *x* to a float
str(x)	Converts *x* to a string

Let's see how we can use these functions to convert number variables in Python.

In [21]:
```
# Converting float to integer
x = 3.14
y = int(x)
print(y)  # Output: 3
print(type(y))
```

```
3
<class 'int'>
```

In [22]:
```
# Converting integer to float
x = 5
y = float(x)
print(y)  # Output: 5.0
print(type(y))
```

```
5.0
<class 'float'>
```

In [23]:
```
# Converting integer to string
x = 10
y = str(x)
print(y)  # Output: '10'
print(type(y))
```

```
10
<class 'str'>
```

In [24]:
```
# Converting float to string
x = 3.14
y = str(x)
print(y)  # Output: '3.14'
print(type(y))
```

```
3.14
<class 'str'>
```

In these examples, we used the int(), float(), and str() functions to convert number variables from one type to another.

Note: It's important to note that when converting a float to an integer, Python will truncate the decimal part of the float. This means that int(3.14) will return 3. If you want to round a float to the nearest integer, you can use the round() function.

In [25]:
```
x = 3.6
y = round(x)
print(y)  # Output: 4
```

```
4
```

Now that we know how to convert number variables in Python, let's move on to some common built-in functions that you can use with number variables.

Common built-in functions

Python provides many built-in functions that you can use with number variables. Here are some of the most common ones:

Function	Description
abs(x)	Returns the absolute value of x
pow(x, y)	Returns x to the power of y
round(x, n)	Returns x rounded to n decimal places
max(x1, x2, ..., xn)	Returns the largest number among x1, x2, ..., xn
min(x1, x2, ..., xn)	Returns the smallest number among x1, x2, ..., xn

Let's see how we can use these functions with number variables in Python.

In [26]:
```
# Finding the absolute value of a number
x = -5
y = abs(x)
print(y)  # Output: 5
```

```
5
```

In [27]:
```
# Finding the power of a number
x = 2
y = pow(x, 3)
print(y)  # Output: 8
```

```
8
```

In [28]:
```
# Rounding a number to a given number of decimal places
x = 3.14159
y = round(x, 2)
print(y)  # Output: 3.14
```

```
3.14
```

In [29]:
```
# Finding the largest number among a list of numbers
x = [5, 10, 2, 8]
y = max(x)
print(y)  # Output: 10
```

```
10
```

In [30]:
```
# Finding the smallest number among a list of numbers
x = [5, 10, 2, 8]
y = min(x)
print(y)  # Output: 2
```

```
2
```

In the above examples, we used the built-in functions to perform common operations on number variables. Python provides many other built-in functions for working with number variables, such as sum(), divmod(), and complex(). You can find more information about these functions in the Python documentation.

Conclusion

By now, we covered a range of topics related to number variables in Python, including their types, how to declare and assign them, and how to perform arithmetic and comparison operations on them. We also explored some common built-in functions that can be used with number variables.

By understanding these concepts, you should now have a good grasp of how to work with number variables in Python, and be able to write programs that perform a wide range of mathematical operations. Remember, practice makes perfect, so be sure to experiment with the examples in this lecture and continue learning more about Python programming.