Number Data Type in Python

Python supports integers, floating point numbers and complex numbers. They are defined as int, float and complex class in Python. Integers and floating points are separated by the presence or absence of a decimal point. 5 is integer whereas 5.0 is a floating point number.

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

We can use the type() function to know which class a variable or a value belongs to and isinstance() function to check if it belongs to a particular class

```
a = 34

# Output: <class 'int'>
print(type(a))

# Output: <class 'float'>
print(type(15.03))

# Output: (3+13j)
cm = 3 + 13j
print(cm)
```

Number system prefix for Python numbers

Number System	Prefix
Binary	'0b' or '0B'
Octal	'0o' or '0O'
Hexadecimal	'0x' or '0X'

```
# Output: 107
print(0b1101011)

# Output: 253 (251 + 2)
print(0xFB + 0b10)

# Output: 13
print(0o15)
```

Type Conversion

We can convert one type of number into another. This is also known as coercion.

Operations like addition, subtraction coerce integer to float implicitly (automatically), if one of the operand is float.

```
>>> 1 + 2.0
3.0
```

We can see above that 1 (integer) is coerced into 1.0 (float) for addition and the result is also a floating point number.

We can also use built-in functions like `int()`, `float()` and `complex()` to convert between types explicitly. These function can even convert from strings.

```
>>> int(2.3)
2
>>> int(-2.8)
-2
>>> float(5)
5.0
>>> complex('3+5j')
(3+5j)
```

When converting from float to integer, the number gets truncated (integer that is closer to zero).

Integer arithmetic

In real life, we often perform arithmetic operations. They help us to calculate the change from a purchase, determine the area of a room, and count the number of people in a line, and so on. The same operations are used in programs.

Basic operations

Python supports basic arithmetic operations:

- addition `+`
- subtraction `-`
- multiplication `*`
- division `/`
- integer division `//`

The examples below show how it works for numbers.

```
print(10 + 10)   # 20
print(100 - 10)  # 90
print(10 * 10)   # 100
print(77 / 10)   # 7.7
print(77 // 10)  # 7
```

There is a difference between division `/` and integer division `//`. The first produces a floating-point number (like `7.7`), while the second one produces an integer value (like `7`) ignoring the decimal part.

Python raises an error if you try to divide by zero.

```
ZeroDivisionError: division by zero
```

Writing complex expressions

Arithmetic operations can be combined to write more complex expressions:

```
print(2 + 2 * 2)  # 6
```

The calculation order coincides with the rules of arithmetic operations. Multiplication has a higher priority level than addition and subtraction, so the operation `2 * 2` is calculated first.

To specify an order of execution, you can use **parentheses,** as in the following:

```
print((2 + 2) * 2)  # 8
```

Like in arithmetic, parentheses can be nested inside each other. You can also use them for clarity.

The minus operator has a unary form that negates the value or expression:

```
print(-10)  # -10
```

```
print(-(100 + 200))  # -300
```

Other operations

The remainder of a division. Python modulo operator % is used to get the remainder of a division. It may come in handy when you want to check if a number is even. Applied to 2, it returns 1 for odd numbers and 0 for the even ones.

```
print(7 % 2)  # 1, because 7 is an odd number
```

```
print(8 % 2)  # 0, because 8 is an even number
```

Here are some more examples:

```
# Divide the number by itself
```

```
print(4 % 4)     # 0
```

```
# At least one number is a float
```

```
print(11 % 6.0)  # 5.0
```

```
# The first number is less than the divisor
```

```
print(55 % 77)   # 55
```

```
# With negative numbers, it preserves the divisor sign
```

```
print(-7 % 3)    # 2
```

```
print(7 % -3)    # -2
```

Taking the remainder of the division by 0 also leads to `ZeroDivisionError`.

Raise a number. Here is a way to raise a number to a power:

```
print(10 ** 2)  # 100
```

This operation has a higher priority over multiplication.

Operation priority

To sum up, there is a list of priorities for all considered operations:

1. parentheses
2. power
3. unary minus
4. multiplication, division, and remainder
5. addition and subtraction

Problems

SIMPLE

1. WAP to find area of a Circle.

2. WAP to find perimeter of a Rectangle.

3. WAP to find the Simple Interest and the total amount when the Principal, Rate of Interest and Time are entered by the user.

4. WAP to convert °C to °F and °F to °C.

5. WAP to find distance between two points.

SELECTION

1. Write a program to find greatest of three numbers

2. Write a program to find whether a given number is even or not.

3. Write a program to whether a given number is prime or not.

4. Write a program to find whether a given string or number is palindrome or not

5. Given 3 sides, WAP to find whether a given triangle is right angled or not.