ITP 115
Programming in Python

Expressions and Operators

Numbers

- Don’t panic! This isn't a math class

- But…sometimes we need to do mathematical operations with numbers (like add, multiply, etc.)

Recall

- Python has two types of variables to store numbers

- `int` `3` `-1` `0` `2011`

- `float` `3.14` `0.094` `-12.0`

- We can do mathematical operations with each

Arithmetic Operations `int` (integers)

Operator	Description	Example	Evaluates To
`+`	Addition	`7 + 3`	`10`
`-`	Subtraction	`7 - 3`	`4`
`*`	Multiplication	`7 * 3`	`21`
`/`	Division (True)	`7 / 3`	`2.333333`
`//`	Division (Integer)	`7 // 3`	`2`
`%`	Modulus	`7 % 3`	`1`
`**`	Exponent	`7 ** 3`	`343`

Arithmetic Operations `float` (real numbers)

Operator	Description	Example	Evaluates To
`+`	Addition	`7.0 + 3.0`	`10.0`
`-`	Subtraction	`7.0 - 3.0`	`4.0`
`*`	Multiplication	`7.0 * 3.0`	`21.0`
`/`	Division (True)	`7.0 / 3.0`	`2.333333`
`//`	Division (Integer)	`7.0 // 3.0`	`2.0`
`%`	Modulus	`7.0 % 3.0`	`1.0`
`**`	Exponent	`7.0 ** 3.0`	`343.0`

General Rules

- With many operations, they work as you would expect

```
4 + 3 → 7
```
int int int

```
4 * 3 → 12
```
int int int

```
4.0 + 3.0 → 7.0
```
float float float

```
4.0 * 3.0 → 12.0
```
float float float

General Rules

- What about combining **`float`** and **`int`**?

`4 + 3.0 → 7.0`

int float float

`4 * 3.0 → 12.0`

int float float

`4.0 + 3 → 7.0`

float int float

`4.0 * 3 → 12.0`

float int float

What about Division?

```
4 / 3 → 1.33333
```

int int float

That's odd…we expected an `int`

```
4 // 3 → 1
```

int int int

Wait? What?

Two Types of Division

- True Division **/**
  - This is what think we usually think of as "division"
  - Result will always be a **`float`** regardless of the input types

- Examples

```
10 / 5          → 2.0
10.0 / 5        → 2.0
4 / 3           → 1.3333
99 / 100.0      → 0.99
```

Two Types of Division

- Integer Division **//**
  - Gives you only the integer part of division
  - **Truncates** (or removes) the decimal part of the answer

```
int   // int     → int
float // float   → float
float // int     → float
```

- Examples

```
10 // 5   → 2     (not 2.0)
4  // 3   → 1     (not 1.333)
99 // 100 → 0     (not 0.9999)
```

The modulo operator (%)

- *Modulo* (modulus – or *mod* for short) gives you the remainder from division
- Example:

  14 divided by 4 is 3 *with a remainder of 2*

  ```
  14 % 4 = 2
  ```

- Uses
  - Determining if an integer is odd or even
  - Determining if one integer is evenly divisible by another integer

Arithmetic operations

- Arithmetic operators only work in pairs
  - Expressions with more than 2 operators are really a series of steps of only 2 operands
  - Results are compounded and used as next operand

```
item1 + item2 + item3 + item4
((item1 + item2) + item3) + item4
```

Operator precedence

- **PEMDAS**

- Can prioritize with parenthesis

```
(cost + tax) * discount
cost + (tax * discount)
```

- Without parentheses, expression are evaluated according to the rules of precedence

Partial List of Operators

Evaluated sooner (higher precedence)

Category	Operators
Parentheses (grouping)	`()`
Exponent	`a**b`
Positive, Negative	`+a, -a`
Multiplication, Division, Modulus	`a * b, a / b, a // b, a % b`
Addition, Subtraction	`a + b , a – b`

Evaluated later (lower precedence)

Specialized assignment operators

- Assignment operators can be combined with arithmetic operators (including `-` `*` `/` `%`)

- Example

```
amount = amount * 2
```

Can be written as

```
amount *= 2
```

Other examples

```
a -= 2
rate /= 59
age += 1
hours %= 24
```