Session Notes — Python & Java Remainder / Modulus

1. Python `math.remainder()`

- `math.remainder(a, b)` gives the **IEEE 754-style remainder**.

Example:

```
import math
math.remainder(10, 4)  # -2.0
10 % 4                  # 2
```

-
- Difference with `%`:
  - `%` is the normal modulus; always positive for positive numbers.
  - `math.remainder()` can be negative; follows rounding rules.

Common error:

```
res = math.remainder(a, b)  # ❌ NameError if math not imported
```

✅ Fix:

```
import math
res = math.remainder(a, b)
```

-

---

2. Java `Math.remainder()` mistake

- `Math.remainder()` **does not exist** in Java.

Using it causes:

```
error: cannot find symbol
```

-
- Correct alternatives:

Integer remainder: use `%`

```
int res = a % b;
```

1.
2. **IEEE floating-point remainder:** use `Math.IEEEremainder(a, b)`

---

3. Java `Math.IEEEremainder()`

- Returns `double`, **not int**.
- Signature: `public static double IEEEremainder(double f1, double f2)`

Example:

```
double res = Math.IEEEremainder(78, 9);  // -3.0
```

-

Formula:

```
IEEEremainder(x, y) = x - y * round(x / y)
```

-
- Can produce **negative results** even for positive numbers.
- **Common errors:**

Assigning to int:

```
int res = Math.IEEEremainder(a, b);  // ❌ incompatible types
```
✅ Fix:

```
double res = Math.IEEEremainder(a, b);
```

  -

4. Difference between `%` and `Math.IEEEremainder()`

Operator	Formula	Example (78, 9)	Notes
`%`	`a - b * floor(a / b)`	6	Normal modulus; easy and intuitive
`Math.IEEEremainder()`	`a - b * round(a / b)`	-3.0	IEEE 754 standard; can be negative

- `%` is **preferred for integers and normal math operations**.
- `Math.IEEEremainder()` is useful only for **scientific/angle normalization computations**.

---

5. Recommendations

- Use `%` for **normal division remainder operations**.
- Use `Math.IEEEremainder()` only for **floating-point calculations** following **IEEE 754 rules**.
- Always match **data types**:
  - `%` → int for integers
  - IEEE remainder → double

Python vs Java Math, Primitives, and In-place Operations: Notes

---

1. Math Operations

Python (`math` module)

- Import required: `import math`

- Common functions:

Function	Description	Example
`math.sqrt(x)`	Square root	`math.sqrt(16)` → `4.0`
`math.pow(a,b)`	Power	`math.pow(2,3)` → `8.0`
`math.floor(x)`	Round down	`math.floor(3.7)` → `3`
`math.ceil(x)`	Round up	`math.ceil(3.1)` → `4`
`math.sin(x)`	Trigonometric	`math.sin(math.pi/2)` → `1.0`
`math.log(x)`	Natural log	`math.log(10)` → `2.302...`
Constants	`math.pi`, `math.e`	

- Built-in functions (no import): `abs()`, `round()`, `max()`, `min()`, `sum()`

Java (`Math` class)

- No import needed (`java.lang` automatically imported)

- Common functions:

Function	Description	Example
`Math.sqrt(x)`	Square root	`Math.sqrt(16)` → `4.0`
`Math.pow(a,b)`	Power	`Math.pow(2,3)` → `8.0`

`Math.floor(x)`	Round down	`Math.floor(3.7)` → `3.0`
`Math.ceil(x)`	Round up	`Math.ceil(3.1)` → `4.0`
`Math.sin(x)`	Trigonometric	`Math.sin(Math.PI/2)` → `1.0`
Constants	`Math.PI`, `Math.E`	

- Example:

```
System.out.println(Math.sqrt(25));
System.out.println(Math.round(3.67));
```

---

2. Primitive vs Object Comparison

Primitives (int, double, char, boolean)

- Use `==` to compare values

```
int a = 10, b = 10;
if(a == b) System.out.println("Equal");
```

Objects (String, Integer, Double)

- Use `.equals()` to compare content

```
String s1 = "Hello", s2 = new String("Hello");
System.out.println(s1.equals(s2)); // true
System.out.println(s1 == s2);      // false
```

- `.equalsIgnoreCase()` → ignores case for strings

```
System.out.println("Hello".equalsIgnoreCase("hello")); // true
```

Rule of Thumb

Type	Compare with	Checks
Primitive	`==`	Value
Object	`.equals()`	Content
String ignore case	`.equalsIgnoreCase()`	Content ignoring case

---

3. Primitive vs Wrapper Classes

Primitive	Wrapper	Notes
int	Integer	object, has methods, can be null
double	Double	object, has methods, can be null
char	Character	object, can be null
boolean	Boolean	object, can be null

Example

Double x = 5.5;
Double y = 2.0;
System.out.println(x + y); // 7.5
System.out.println(x.equals(y)); // false

- **Autoboxing**: automatic conversion between primitive and wrapper

Double obj = 10.5; // primitive double -> Double
double num = obj;   // Double -> primitive double

---

4. Python vs Java: In-place list/array modifications

Python

```
arr = [1, 2, 3]
for i in range(len(arr)):
    arr[i] += 5  # modifies original list
print(arr)  # [6, 7, 8]
```

- Lists are mutable, integers are immutable.
- `.append()` adds elements to list, cannot call on int.

Java

```
int[] arr = {1,2,3};
for(int i = 0; i < arr.length; i++) {
    arr[i] += 5;
}
System.out.println(Arrays.toString(arr)); // [6, 7, 8]
```

- Arrays are mutable, primitives cannot call methods.
- ArrayList allows `.add()` and `.set()`.

---

5. Python does not support `++` / `--`

- Java:

```
i++;  // increments by 1
++i;  // pre-increment
```

- Python:

```
i += 1  # increment
i -= 1  # decrement
```

- Reason: Python favors explicit, readable operations.

---

6. Python Example: In-place multiples of 5

```
n = int(input("Enter n: "))
arr = [i+1 for i in range(n)]

def keep_multiples_of_five(lst):
    i = 0
    while i < len(lst):
        if lst[i] % 5 != 0:
            lst.pop(i)
        else:
            i += 1

keep_multiples_of_five(arr)
print("Multiples of 5:", arr)
print("Sum:", sum(arr))
```

- Modifies `arr` in-place, no new list needed.

End of Notes