# ENGF0002 (Design and Professional Skills)

## Scenarios

The focus of this document is on the differences in number representation between the three prototypes of the programming language. It follows an observation-explanation-conclusion structure in that an observation is laid down, explained, and a theory is formulated it out of it.

Classifier-1

```
#lang Numbers

block:
  a = 2
  b = 3
  print( a / b)
end

block:
  a = 4
  b = 3
  print( a / b)
end

block:
  a = 22
  b = 7
  print( a / b)
end

block:
  a = 3.0
  b = 7
  print( a / b)
end
```

```
Language: Numbers, with debugging; memory limit: 128 MB.
version: 2018-09-04T22:54:09-04:00

------Core 1------

0.6666666667
1.3333333333
3.1428571429
0.4285714286

------Core 2------

0.6666666667
1.3333333333
3.1428571429
0.4285714286

------Core 3------

0
1
3
0.4285714286
```

Observation-1: When two integers are divided- with a decimal result- then Core-1 and 2 represent the result upto 10 decimal places, while Core-3 rounds the answer to the closest integer.

Theory-1: This indicates that when two integers are divided, Core-3 represents the result as an integer, whereas Core-1 and 2 represent the result as a double/float. Since all the prototypes are dynamically typed (type conversion is implicit) this shows that Core-1 and 2 treat the result of a numeric operation separately and convert it to another type, if need be. However, Core-3 binds the result to the type of the divisor and the dividend. *Example- 22/7 = 3.1428571429 (core 1,2) = 3 (core 3)*

*Partition after test: {Core 1, Core 2}, {Core 3}*

This raises an important question: what does Core-3 do when a floating-point number is divided by an integer?

Observation-2: When an integer and a real number are divided, all three languages represent the result as a real number.

Theory-2: This shows that all three languages represent the value as a double/float if an integer and a double value are divided; indicating that Core-3 treats the result as a double if a double value is included in the operation and as an integer if only integer division is involved. *Example- 3.0/7 = 0.4286714286 (core 1,2,3)*
*Partition after test: {Core 1, Core 2, Core 3}* (no partition)

Classifier-2

After testing the representation of integer division, next, I tested the representation of big or "long" integers.

```
#lang Numbers
print(1024 * 1024 * 1024 * 1024 * 1024 * 1024)
print(1024 * 1024 * 1024 * 1024 * 1024 * 1024 * 2)
print(1024 * 1024 * 1024 * 1024 * 1024 * 1024 * 4)
print(1024 * 1024 * 1024 * 1024 * 1024 * 1024 * 8)
print(9999999999999999999)
print(999999999999999999)
```

```
Welcome to DrRacket, version 7.0 [3m].
Language: Numbers, with debugging; memory limit: 128 MB.
version: 2018-09-04T22:54:09-04:00

------Core 1------

1152921504606846976
2305843009213693952
4611686018427387904
9223372036854775808
9999999999999999999
999999999999999999

------Core 2------

1152921504606846976
2305843009213693952
4611686018427387904
9223372036854775808
10000000000000000000
1000000000000000000

------Core 3------

1152921504606846976
2305843009213693952
4611686018427387904
-9223372036854775808
-8446744073709551617
999999999999999999
>
```

Observation-3: When any number bigger than $2^{63} - 1$ is printed, Core-3 prints out ambiguously; printing negative integers without any relation to the number. Core-1, on the other hand, correctly prints out the number.
*Example- when $1024^6$ or $2^{60}$ is printed out, all the prototypes print out correctly- similar to when $2^{61}$, $2^{62}$ and $2^{63} - 1$ are printed.*
However, the moment a number exceeds $2^{63} - 1$, the behaviour of the prototypes differ widely from each other.

Theory-3: This implies that the long integer type in Core-3 has a maximum limit of $2^{63} - 1$, which makes it behave incorrectly whenever a bigger number than that is being used.
Example- *$(1024^6 * 8)$ OR $2^{63}$ prints out as $-9223372036854775808$. However, $(1024^6 * 8 - 1)$ prints out correctly.*

For Core-2, after multiple investigations, I found that any number that is greater than $2^{52}$ with multiple 9s at the end prints out as the next biggest integer.
*Example- 99999999999999999 prints out as 1e18.*
*This is, however, only the case with numbers with multiple 9s at the end as other numbers that are greater than $2^{52}$ are printed out correctly*

Core-1, however, behaves normally with long integers as any integer, regardless of its length is printed out correctly; showing durability when it comes to maintaining accuracy.

*Partition after test: {Core 1}, {Core 2}, {Core 3}*

Classifier-3

Finally, I checked to see the difference in error-representation, specifically a run-time error, by dividing numbers by zero.

```
#lang Numbers

block:
  a = 20.462
  b = 0
  print(a / b)

  a = 20
  print(a / b)
end
```

```
Welcome to DrRacket, version 7.0 [3m].
Language: Numbers, with debugging; memory limit: 128 MB.
version: 2018-09-04T22:54:09-04:00

------Core 1------

Inf
Inf

------Core 2------

Inf
Inf

------Core 3------

Inf
ERROR: Cannot divide by zero.

>
```

Observation-4: When any number is divided by zero, Core-1 and 2 print the result as *Inf* (or Infinity) Core-3 does the same when a floating-point number is divided by zero. However, when an integer is divided by zero, Core-3 prints an error, unlike Core-1 and 2 which still print *Inf*.
*Example: 20 / 0 = Inf (core 1,2) = ERROR: Cannot divide by zero (core 3)*

Theory-4: This implies that when an integer is divided by zero, Core-3 will report an error and not compile. However, Core-1 and 2 will return an *Inf* value which means that the program will compile. However, none of the prototypes report an error when a floating-point number is divided by zero as they all return *Inf*.
*Example:*
*20.462 / 0 = Inf (core 1,2,3)*

*Partition after test: {Core 1, Core 2}, {Core 3}*