09-11 (Handout)

1 Retrieval

2 Python Calculator (POGIL, 15 min)

In a Python Shell window, `>>>` is a *prompt* indicating that the interpreter is waiting for input. All text entered after the prompt will be executed immediately as Python code.

If you type a Python *expression* (code that results in a value) after the prompt, Python will show the value of that expression, similar to a calculator. You can use Python's `math` module to perform more complex mathematical operations like logarithms and trigonometric operations.

Do not type anything yet! Read the questions first!

Python code	Predicted output	Actual output
`2 + 3`		
`3 * 4 + 2`		
`3 * 4 + 2.0`		
`3(4 + 2)`		
`3 * (4 + 2)`		
`5 / 10`		
`5 / 10.0`		
`5 / 9`		
`2 ** 4`		
`abs(-2) ** 4`		
`math.pow(2, 4)`		
`import math`		

Python code	Predicted output	Actual output
`math.pow(2, 4)`		
`sqrt(4)`		
`math.sqrt(4)`		
`math.cos(0)`		
`math.pi`		
`math.sin(math.pi / 2)`		

1. In the middle "Predicted output" column, write what value you expect will be displayed, based on your team's experience using a calculator. If there are any lines you are not confident about, place an asterisk next to your predicted output.
2. Open a Python Shell on your computer. Type each Python expression at the prompt, one line at a time, and write the corresponding Python output in the third column above. If an error occurs, write what type of error it was (i.e., the first word of the last line of the error message).
3. What does the `**` operator do?

4. Based on the Python code in the expressions table, identify four examples of:
   - Mathematical operators:
   - Mathematical functions:
5. For addition and multiplication to produce an output with a decimal value, what type of number must be part of the input? Provide justification for your team's answer.

6. Does division follow the same rule as in the previous question? Provide justification for your team's answer.

1. The output of the expressions table displayed three different errors. Explain the reason for each:

- `TypeError`:
- 1st `NameError`:
- 2nd `NameError`:

2. Identify two differences between using a Python built-in function (e.g., `abs`) and a function from the `math` module.

3 Dividing Numbers (POGIL, 15 min)

9 / 4	*evaluates to*	2.25
10 / 4	*evaluates to*	2.5
11 / 4	*evaluates to*	2.75
12 / 4	*evaluates to*	3.0
13 / 4	*evaluates to*	3.25
14 / 4	*evaluates to*	3.5
15 / 4	*evaluates to*	3.75
16 / 4	*evaluates to*	4.0

9 // 4	*evaluates to*	2
10 // 4	*evaluates to*	2
11 // 4	*evaluates to*	2
12 // 4	*evaluates to*	3
13 // 4	*evaluates to*	3
14 // 4	*evaluates to*	3
15 // 4	*evaluates to*	3
16 // 4	*evaluates to*	4

9 % 4	*evaluates to*	1
10 % 4	*evaluates to*	2
11 % 4	*evaluates to*	3
12 % 4	*evaluates to*	0
13 % 4	*evaluates to*	1
14 % 4	*	

evaluates to* | 2 | | 15 % 4 | *evaluates to* | 3 | | 16 % 4 | *evaluates to* | 0 |

1. For each operator in the tables, identify the symbol and describe the type of numerical result.

2. If the result of the `/` operator were rounded to the nearest integer, would this be the same as the result of the `//` operator? Explain how the results in Table~A compare to Table~B.

3. If the table included more rows, list all numbers `// 4` would evaluate to 2 and all the numbers `// 4` would evaluate to 4.

4. Based on the results of Table~C, propose another number `% 4` evaluates to 0, and explain what all these numbers have in common.

5. Consider the expressions in Table~C that evaluate to 1. How do the left *operands* in these expressions (i.e., 9, 13) differ from those that evaluate to 0?

6. Describe the reason for the repeated sequence of numbers (0, 1, 2, 3) for the result of `% 4`.

7. Imagine that you are given candy mints to divide evenly among your team members.

- If your team receives 11 mints, how many mints would each student get, and how many are left over? Write a Python expression to compute each result.

- If your team receives 2 mints, how many mints would each student get, and how many are left over? Write a Python expression to compute this result.

8. Python has three division operators: "floor division", "remainder", and "true division". Which operator (symbol) corresponds to each name?

4 Operators

4.1

Operation	Operator	Example	Evaluates to
Addition	`+`	`2 + 2`	`4`
Subtraction	`-`	`4 - 1`	`3`
Multiplication	`*`	`1.5 * 2`	`3.0`
Division	`/`	`5 / 2`	`2.5`
Floor Division	`//`	`5 // 2`	`2`
Modulus (remainder of division)	`%`	`5 % 2`	`1`
Exponent	`**`	`3**3`	`27`

4.2 Resulting types

- If any of the operands is a float, result will be a float. Otherwise (both are integer), result is an integer.
- However, there is an exception: result of a **division** (not floor division) is ***always*** a float. Careful with that! (why? we'll see in a moment)

4.3 Operator precedence

- Always be careful with expressions using more than one operator! For example:

```
a = 3
b = 6
c = a + b * 2
print(c)
```

- This evaluates as $a + (b \times 2) = 3 + (6 * 2) = 3 + 12 = 15$

Python operator **precedence order**:

1. Parentheses: `()`
2. Exponents: `**`
3. Multiplication, divisions and modulus: `* / // %`
4. Addition and subtraction: `+ -`
5. Comparisons: `<= < >= > == != is` (next week)
6. Boolean `not` (next week)
7. Boolean `and` (next week)
8. Boolean `or` (next week)

4.4 Operators in strings

- Python also permits using SOME operators with strings. In a metaphorical way…

String Operation	Metaphor	Operator	Example	Evaluates to
Concatenation	Addition	`+`	`"Hey" + " " + "apple"`	`"Hey apple"`
Repetition	Multiplication	`*`	`"na" * 4`	`"nananana"`

- Other operators are not supported. Multiplication of a string with another string is also not supported. Both wouldn't make so much sense…

4.5 Evaluating operations in strings

It is also possible to evaluate an expression coded as a string. For example:

```
expression = "2 * (4 + 6) / 3 - 5"
result = eval(expression)
print(result)
```

```
1.666666666666667
```

5 Math module

5.1

For other mathematical operations, you can import Python's math module. Two ways to do that:

- Import the whole module and use as needed: need to use the module name followed by a dot (.) before using a function or variable from the module.

```
import math
r = 3
circumference = 2 * r * math.pi
```

- Import only the functions and variables you will need. Then, no need to use the qualifier "math."

```
from math import pi
r = 3
circumference = 2 * r * pi
```

Some functions in the math module (documentation here)

Method	Explanation
math.cos(x)	Returns cosine of x (rads)
math.sin(x)	Returns sine of x (rads)
math.tan(x)	Returns tangent of x (rads)
math.degrees(x)	Converts x radians to degrees
math.radians(x)	Converts x degrees to rads
math.sqrt(x)	Returns square root of x

- You can also try to import all of the modules resources by using

```
from math import *
```

- However, this is not recommended due to **"namespace pollution"** (having lots of names referenced that are actually not used)

6 Overflow

6.1

- Integers and floats are representations in the memory of your computer. Therefore, there are value limits to these numbers.
  - If we pass the limit, we arrive at an **overflow**, and the number is not computed correctly.

- **For integers**, Python dynamically increases the use of memory as the number grows. Therefore, **in theory** there is a limit (computer memory will be full), but **in practice** this will almost never be the case.
  - For example, try `2147483647 ** 200`
  - This, however, is a Python feature. Other languages may not handle that.

- **For floats**, numbers are limited by the size of the mantissa and exponent.
  - Remember: for 1.2345×10^7, 1.2345 is the mantissa, and 7 is the exponent
  - In Python, maximum is about 3 digits for the exponent and 17 digits for the mantissa
  - For example, try: `2.0**2000`
    - `OverflowError`

More info at https://docs.python.org/3/tutorial/floatingpoint.html

6.2 So, be careful when mixing numeric types!

- On June 4, 1996 the European Space Agency launched the first Ariane 5 rocket:
  - the result of a decade of development, $8 billion
  - Exploded 40 seconds after lift-off with a $500 million satellite payload on board…
- Cause was a software error related to a **number type conversion and an overflow**

http://www-users.math.umn.edu/~arnold/disasters/ariane.html