# Introduction to Python II

Today, we'll continue learning about the programmable internal-state machines, in particular, about order of operations, built-in functions, and data types.

## Warm-up

Last class, we looked at some basic instructions, namely ***arithmetic expressions*** and ***variable assignment***. Test your knowledge by answering the following question:

## Questions (15 min) **Start time:**

**1**. In the "Result" column, write what value you expect will be the result and indicate the type of the numerical result:

| Python code | Result |
|---|---|
| `6/3` | 2.0 (decimal / float) |
| `5/2` | 2.5 (decimal / float) |
| `5//2` | 2 (whole / integer) |
| `5%2` | 1 (whole / integer) |
| `5%3` | 2 (whole / integer)) |
| `6%2` | 0 (whole / integer) |
| `6%3` | 0 (whole / integer) |
| `5**2` | 25 (whole / integer) |
| `5 * 2` | 10 (whole / integer) |
| `5 * 2.0` | 10.0 (decimal / float) |
| `5.0 * 2` | 25.0 (decimal / float) |
| `5 * 2.0 + 1` | 11.0 (decimal / float) |
| `(1.0 + 7)//2` | 4.0 (decimal / float) |

**2**. Explain why `x=y` and `y=x` are different in computer science.

Equality in math represents a relation that is two way; for that reason in math `x=y` and `y=x` are equivalent, but they are NOT in computer science. `=` represents assignment in computer science and it works one way: the value on the right-hand-side is copied to the variable on the left-hand-side of the operator.

**3**. Show how the Python Machine interprets and executes these code statements:

| Python code | Output |
|---|---|
| `x = 3` | |
| `print("x=",x)` | x= 3 |
| `x = 5` | |
| `print("x=",x)` | x= 5 |
| `y = x + 2` | |
| `print("y=",y)` | y= 7 |
| `y = y + x` | |
| `print("y=",y)` | y= 12 |
| `print("x=",x)` | x= 5 |

| Interpreter | Basic Instructions Pad | State |
|---|---|---|
| Assignment | x=3 | x:3 |
| Read | Replace x with 3 | x:3 |
| Function: print | print("x=",3) | x:3 |
| Assignment | x=5 | x:5 |
| Read | Replace x with 5 | x:5 |
| Function: print | print("x=",5) | x:5 |
| Read | Replace x with 5 | x:5 |
| Addition | 5+2=7 (value not saved) | x:5 |
| Assignment | y=7 | x:5 \| y:7 |
| Read | Replace y with 7 | x:5 \| y:7 |
| Function: print | print("y=",7) | x:5 \| y:7 |
| Read | Replace y with 7 | x:5 \| y:7 |
| Read | Replace x with 5 | x:5 \| y:7 |
| Addition | 7+5=12 (value not saved) | x:5 \| y:7 |
| Assignment | y=12 | x:5 \| y:12 |
| Read | Replace y with 12 | x:5 \| y:12 |
| Function: print | print("y=",12) | x:5 \| y:12 |
| Read | Replace x with 5 | x:5 \| y:12 |
| Function: print | print("x=",5) | x:5 \| y:12 |

# Model 1 Order of Operations

Python follows a specific order for math and other operations. For example, multiplication and division take ***precedence*** over addition and subtraction. The following table lists several Python operators from highest precedence to lowest precedence.

| Operator | Description |
|---|---|
| `**` | Exponentiation |
| `+ -` | Positive, Negative (***unary*** operators) |
| `* / // %` | Multiplication, Division and Modulus |
| `+ -` | Addition, Subtraction (***binary*** operators) |
| `=` | Assignment |

## Questions (10 min) **Start time:**

**4**. Determine the order of operations in the statement: `y = 9 / 2`

a) First operator to be evaluated: `/`

b) Second operator: `=`

c) Value of y: 4.5

**5**. Determine the order of operations in the statement: `x = 5 * -3`

a) First operator to be evaluated: `-`

b) Second operator: `*`

c) Third operator: `=`

d) Value of x: -15

**6**. Determine the order of operations in the statement: `z = 2 * 4 ** (3 + 1)`

a) First operator to be evaluated: `+`

b) Second operator: `**`

c) Third operator: `*`

d) Fourth operator: `=`

e) Value of z: 512

**7**. The + and - operators show up twice in the table of operator precedence. For the Python statement `x = 5 * -3`, explain how you know whether the - operator is being used as a unary or binary operator.

It matters what is to the left or right of an operator. In this example, the - is preceded by a *, so it must be unary.

8. What do the words "unary" and "binary" mean in this context?

Unary means there is a single operand (one value to operate on), and binary means there are two operands (left and right).

9. Evaluate the following expressions. Why are the results different? Explain your answer in terms of operator precedence.

- `-3 ** 2` Result: -9
- `(-3) ** 2` Result: 9

Exponentiation has a higher precedence than unary operators. So in the first expression, 3 is squared before the entire value is negated.

## Model 2 Python Built-In Functions

Recall that in addition to ***operator expressions*** we also have ***function-call expressions***. You can use ***functions*** to perform specific operations. Some functions require values, known as ***arguments***, to perform their operation. Functions may also ***return*** a result. For example:

```
name = input("What's your name? ")
```

`input` is a function, `"What's your name? "` is an argument, and the return value (typed by the user) is stored in `name`. Python has a list of functions that are always available (called built-in). See https://docs.python.org/3/library/functions.html for a complete list and the back of the handout for a course-specific list.

### Questions (15 min) Start time:

10. Evaluate each code statement below and write down the output:

| Python code | Output |
|---|---|
| `input("enter the mass in grams:  ")` | enter the mass in grams: 100 |
| `mass = input("enter another mass in grams:  ")` | enter another mass in grams: 10 |
| `print(mass)` | '10' |
| `ten = 10` | |
| `print(ten / 2)` | 5.0 |
| `abs(-1)` | 1 |
| `abs(-1 * ten)` | 10 |

**11**. List the names of the three functions used above. input, print, abs

**12**. What are the arguments of the first use of the `print` function? mass, unit

**13**. Which function delays execution until additional input was entered? The input function.

**14**. Which term, ***user*** or ***programmer***, best defines the role of the person who entered the additional input? Explain.

User is a better term; programs are ultimately written to be used by non-programmers.

**15**. What does the word `mass` represent, and how did it get its value?

(Answers may vary) Its value is 10, which the user entered as input.

**16**. What does the word `ten` represent, and how did it get its value?

Its value is 10 based on the statement "ten = 10".

**17**. Do the values of `mass` and `ten` both represent a number? Explain why or why not.

Although they both appear to represent a numerical value, `mass` divided by 2 gives an error, while `ten` divided by 2 gives the expected numerical outcome.

# Model 3 Integers and Floats

Every value in Python has a ***data type*** which determines what can be done with the data.

## Questions (10 min) Start time:

**18**. Evaluate the following code statements and write down the output:

| Python code | Output |
|---|---|
| `integer = 3` | |
| `pi = 3.1415` | |
| `word = str(pi)` | |
| `print(word)` | '3.1415' |
| `number = float(word)` | |
| `print(word * 2)` | 3.14153.1415 |
| `print(number * 2)` | 6.283 |
| `print(number + 2)` | 5.14159 |
| `euler = 2.7182` | |
| `print(int(euler))` | 2 |
| `print(round(euler))` | 3 |

**19**. What is the data type (`int`, `float`, or `str`) of the following values?

a) pi float

b) integer int

c) word str

d) number float

**20**. List the function calls that convert a value to a new data type.

The calls are: `str(pi)`, `float(word)`, and `int(euler)`. Note there is a function named after each data type.

**21**. How does the behavior of the operators (+ and *) depend on the data type?

The + operator appends text, and the * operator copies text.

# Model 4 Errors

Errors signal bugs in our program. Bugs can be due to writing instructions improperly (**syntax errors**), improper instructions or ordering (**runtime errors**) or incorrect output (**semantic errors**).

## Questions (15 min) **Start time:**

**22**. Write the corresponding output for each statement assuming that they are executed in order. If an error occurs, write what type of error. Place an asterisk (*) next to any output for which you are unsure.

| Python code | Output |
|---|---|
| `data = 12` | |
| `print(data)` | 12 |
| `print(Data)` | NameError |
| `Data = input("Type input:")` | |
| `print(data)` | 12 |
| `print(Data / 2)` | Type Error |
| `my data = 56` | SyntaxError |
| `my_data = 78` | |
| `3data = "hello"` | SyntaxError |
| `data3 = "world"` | |
| `data3 = hello` | NameError |
| `mass = 273 + 100` | |
| `273 + 100 = mass` | SyntaxError |
| `print(mass)` | 373 |
| `Mass + 100` | NameError |
| `result = 3(2+4)` | TypeError |

**23**. Indicate whether each statement below is true or false.

a) Variable names in Python can start with a number. false

b) Variable names in Python must start with a lower-case letter. false

c) Variable names in Python may not include spaces. true

d) Variable names in Python are case-sensitive. true

**24**. Each of the following assignment statements has an error. Write a valid line of Python code that corrects the assignment statement. Double-check your code using a computer.

a) `3 + 4 = answer` answer = 3 + 4

b) `oh well = 3 + 4` oh_well = 3 + 4

c) `2x = 7` x2 = 7

## Built-in Functions Cheatsheet

| Function | From Python's Docs: https://docs.python.org/3/library/functions.html |
|---|---|
| `abs(x)` | Returns the absolute value of a number. The argument may be an integer or a floating point number. |
| `chr(i)` | Returns the string representing a character whose Unicode code point is the integer i. For example, chr(97) returns the string 'a', while chr(8364) returns the string '€'. This is the inverse of ord(). |
| `float(x=0.0)` | Returns a floating point number constructed from a number or string `x`. |
| `input(prompt)` | Reads a line from input, converts it to a string (stripping a trailing newline), and returns that. |
| `int(x, base=10)` | Returns an integer number constructed from a number or string `x`, or return 0 if no arguments are given. |
| `len(s)` | Returns the length (the number of items). The argument may be a sequence (such as a string, list, or range) or a collection (such as a dictionary). |
| `max(arg1, arg2, *args, key=None)` | Returns the largest of two or more arguments. |
| `min(arg1, arg2, *args, key=None)` | Returns the smallest of two or more arguments. |
| `ord(c)` | Given a string representing one Unicode character, return an integer representing the Unicode code point of that character. For example, ord('a') returns the integer 97 and ord('€') (Euro sign) returns 8364. This is the inverse of chr(). |
| `pow(base, exp)` | Returns `base` to the power `exp`; it is equivalent to using the power operator: `base**exp`. |
| `print(*objects, sep=' ', end='\n', file=None, flush=False)` | Prints to the text stream file, separated by `sep` and followed by `end`. |
| `round(number, ndigits=None)` | Returns `number` rounded to `ndigits` precision after the decimal point. If `ndigits` is omitted or is `None`, it returns the nearest integer to its input. |
| `str(object='')` | Returns a str version of object. |
| `type(object)` | Returns the type of an object. |