Lecture 1

January 19, 2023

1 Value and Type

- Value is a basic thing a program works with, like a number: 2, 4.0; or a letter “o”; or a sentence “Hello MATH 183”.
- These values belong to different types:
- 2 is an integer,
- 4.0 is a floating-point number,
- ‘Hello MATH 183’ is a string, so-called because the letters it contains are strung together
- what data type is “o”?

[38]:
```
o = 'o'
print(o)
print(type(o))

a= 2
print(a) #value
print(type(a))  #type

b= "2.0"
print(b)
print(type(b))

c= "Hello MATH 183"
print(c)
print(type(c))
```

```
o
<class 'str'>
2
<class 'int'>
2.0
<class 'str'>
Hello MATH 183
<class 'str'>
```

2 Programming Language

Programming languages are **formal languages** that are designed to express computations.

Formal languages tend to have strict syntax rules that govern the structure of state- ments. For example, in mathematics:

- "$3 + 3 = 6$" has correct syntax,
- "$3+ = 3\$6$" does not.

3 Syntax rules

Syntax has two parts: 1) tokens, and 2) structure

- Tokens are the **basic elements** of the language, such as words, numbers, and chemical elements. One of the problems with "$3+ = 3\$6$" is that "$" is not a legal token in mathematics.
- Structure is the **way tokens are combined**. The equation "$3+ = 3$" is illegal because even though "+" and "=" are legal tokens, you can't have one right after the other.

4 Variable and Assignment

One of the most powerful features of a programming language is the ability to manipulate variables. A **variable** is a **name** that refers to a value.

An assignment statement creates a new variable and gives it a value:

```
message = 'And now for something completely different'
n = 17
pi = 3.141592653589793
```

This example makes three assignments: - The first assigns a string to a new variable named message; - the second gives the integer 17 to n; - the third assigns the (approximate) value of π to pi.

5 Variable Names

- We choose names for variables that are meaningful—they document what the variable is used for.
- Variable names can be as long as you like. They can contain both letters and numbers, but they can't begin with a number. It is legal to use uppercase letters, but it is conven- tional to use only lowercase for variables names.
- The underscore character, _, can appear in a name. It is often used in names with multiple words, such as "your_name" or "airspeed_of_unladen_swallow".

6 Name Errors

If you give a variable an illegal name, you get a syntax error:

```
>>> 76trombones = 'big parade'
SyntaxError: invalid syntax
>>> more@ = 1000000
SyntaxError: invalid syntax
>>> class = 'Advanced Theoretical Zymurgy'
SyntaxError: invalid syntax
```

- 76trombones is illegal because it begins with a number.
- more@ is illegal because it con- tains an illegal character, @.

But what's wrong with class?

7 Keywords

- Reason: "class" is one of Python's **keywords**. The interpreter uses keywords to recognize the structure of the program, and they cannot be used as variable names.
- Python 3 has these keywords: False; True; class; finally;
- In most development environments, keywords are displayed in a different color; if you try to use one as a variable name, you'll know.

```
class = 10
my_variable= 10
```

8 Built-in Functions

We can use built-in functions (possibly from a package) or we can define a function.

Example: The package "math" has a of built-in functions, such as "log", "exp", sin. We can combine them together:

- $\tan(1)$,
- $\sin(\tan(1)) + \log(10)$.

[25]:
```
import math

math.tan(1)
math.sin(math.tan(1))+ math.log(10)
```

[25]:
```
3.3024954669992495
```

9 Hand-crafted Functions

We can define a function on our own, normally in the following situations:

- we want to repeat a task over and over
- we want to define an output that depends on some output(s).

See below examples.

[51]:
```
def my_function():
    # This function print out 'Hello MATH 183'
    print('Hello MATH 183')

# Call my_function
my_function()

def my_double_log(a):
    # This function return 2*log(a)
    return 2*math.log(a)

# Call my_double_function
# You can change the input into anything that you want

my_double_log(1)
```

```
Hello MATH 183
```

[51]:
```
0.0
```

10 Practice problem

Problem: Write a program to add all integers from 1 to 1000?

Solution: We want to compute $1 + 2 + \cdots + 1000$.

Method 1: use Math formula

$$1 + 2 + \cdots + 1000 = \frac{n(n+1)}{2}.$$

Method 2: add one element each time to the sum, that is:

- Step 0: $s = 0$
- Step i: $s \to s + i$, for $i = 1, \cdots, 1000$.

[48]:
```
#Code for Method 1
n = 1000
print(n*(n+1)/2)


# Cod for Method 2
s = 0
for i in range(1000):
    s += i+1
print(s)
```

```
500500.0
500500
```

11 General problem

We want to add $1 + 2 + \cdots + n$, where n is a given numner. Then, we should write a **function**, as below.

[50]:
```
def sum_up_to_n(n):
    s = 0
    for i in range(n):
        s += i+1
    print(s)

# Call the function to calculate the sum
# You can use any value for n
sum_up_to_n(1000)
sum_up_to_n(100)
```

```
500500
5050
```

12 Check-out ticket

Write down the program to compute $1^2 + 2^2 + \cdots + n^2$, where n is a given integer.