### CSC108 Notes Template 2 - Basics

# 1 Python as a calculator

Let's start with some basic operators:

Symbol	Operation	Expression	English description	Value
+	addition	11 + 56	11 plus 56	67
-	subtraction	23 - 52	23 minus 52	-29
*	multiplication	4 * 5	4 times 5	20
**	exponentiation	2 ** 5	2 to the power of 5	32
/	division	7/3	7 divided by three	2
%	remainder	7 % 3	7 mod 3	1

**Q.** With  $7 \neq 3$  we only get the quotient and with  $7 \approx 3$  we only get the remainder. What if we want "real" division?

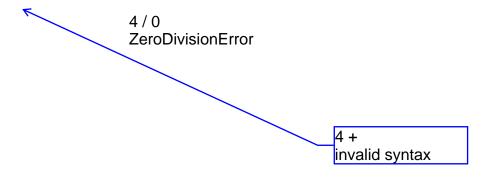
**A.** 7.0/3.0 or 7.0/3 or 7/3.0

# floating point numbers are approximations to real numbers.

Expressions with multiple operators:

Negative numbers:

**Q.** What happens when we try to evaluate an expression that can't work (e.g., 4 / 0)?



#### 

```
Two useful functions: dir and help

See the related Doc.

See the related Doc.

You can use the built-in dir function to list the identifiers that a module defines. The identifiers are the functions, classes and variables defined in that module.
```

#### 3 Variables

Sometimes you need to compute a value that you will use several times:

bio\_calc.py

```
1. print "My average on the BIO test was", 87.5 / 112 * 100
2. if 87.5 / 112 * 100 >= 90:
3.    print "Awesome"
4. elif 87.5 / 112 * 100 >= 70:
5.    print "Doing fine"
6. elif 87.5 / 112 * 100 >= 50:
7.    print "So so"
8. elif 87.5 / 112 * 100 < 50:
9.    print "Uh oh!!"</pre>
```

A variable is a name that refers to a value. Let's improve our bio\_calc.py by introducing a variable:

bio\_calc\_better.py

```
my_average = 87.5/112*100
print "My average on the BIO test was", my_average
if my_average>=90:
    print "Awesome!"
elif my_average>=70:
    print "Doing fine"
elif my_average>=50:
    print "So so"
elif my_average<50:
    print "Uh oh!!"

[actually "my_average" does not stand for a certain number but the location where this number is stored.
```

#### Form of an assignment statement:

variable = expression

variable name always on the RHS!!! otherwise will be an error

How its executed

- 1. Evaluate the expression on the RHS. (The value of the expression is a memory address.)
- 2. Store that memory address in the variable on the LHS.

### 4 Functions

#### 4.1 Defining a function

Defining and using a function in math:

 $f(x)=x^2$ 

 $f(3)=3^2=9$ 

 $f(2)=2^2=4$ 

Defining a function in Python: the function "def"

def f(x): return x \*\* 2

x is the parameter

after using the function "return", type in f(3) in the shell then we will get a 9!

for f(3), 3 is an argument

#### Form of a function definition:

def function\_name(parameters):
 body

def: a Python keyword

parameters: 0 or more parameters, comma separated

body: 1 or more statements

## 4.2 Calling a function

So far, we have just defined what f is; what it means to "do" ("execute") f. We haven't said to do it. When we want Python to execute a function, we "call" it.

Example:

#### Form of a function call:

```
function_name(arguments):
```

How it's executed:

- 1. Each argument is an expression. Evaluate these expressions, in order. (The value of each expression is a memory address.)
- 2. Store those memory addresses in the corresponding parameters.
- 3. Execute the body of the function.

## 5 Tracing

Let's trace the following code using Wing and by hand:

```
    def f(x):
    result = x ** 2
    return result
    size = 11
    bigger = f(size)
```

# 6 Types

We said that a variable is a name that refers to a value. Variables also have a type: the kind of value they are holding.

Let's use Python's function type to find out the types of the numbers we've been working with:

also can use type function to find the type of the variable!

## 6.1 Type conversion

There are also functions that take a value of one type and "convert" it to another:

```
ps:
>>>8e+2 # 8.0*10**2
800.0
```