Numbers

Section 1

Chapter 2

Agenda

- Quiz 2 (on Slides_1 concepts)
- Arithmetic operations
- Object types
- Variables
- Error types
- Lab 2

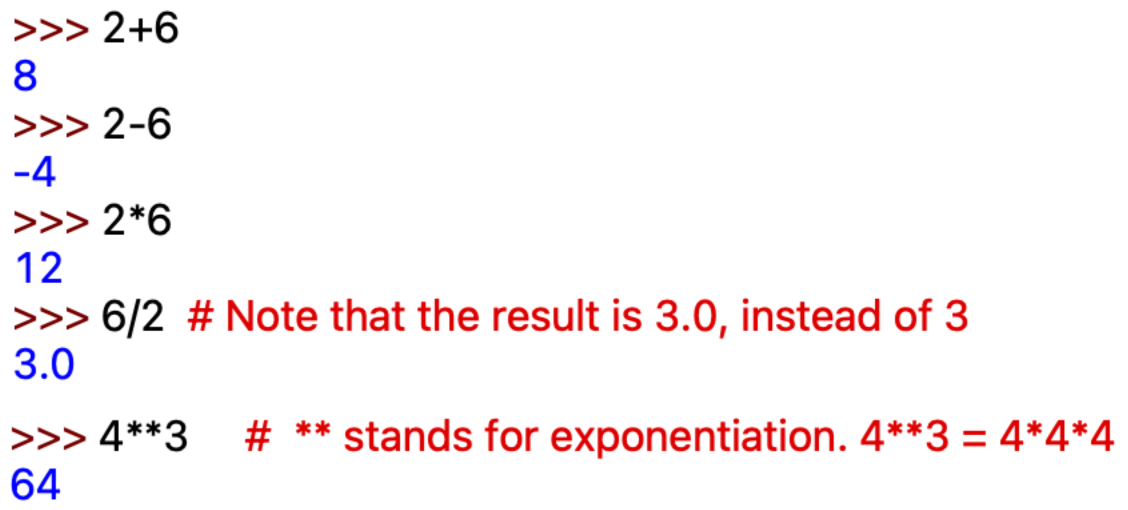
Numbers

- Numbers are referred to as *numeric literals*

- Two Types of numbers
  - Number without a decimal point is called an **int**
  - Number with a decimal point is called a **float**

Arithmetic Operators

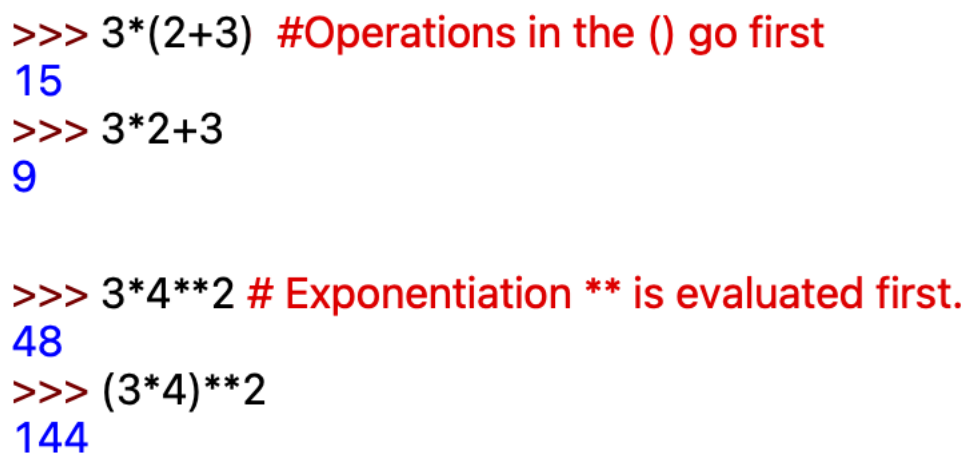
- Addition
- Subtraction
- Multiplication
- Division
- Exponentiation (a^b)



```
>>> 2+6
8
>>> 2-6
-4
>>> 2*6
12
>>> 6/2  # Note that the result is 3.0, instead of 3
3.0
>>> 4**3     #  ** stands for exponentiation. 4**3 = 4*4*4
64
```

Order of Precedence

- Order of precedence
  1. Terms inside parentheses (inner to outer)
  2. Exponentiation
  3. Multiplication, division (ordinary and integer), modulus
  4. Addition and subtraction

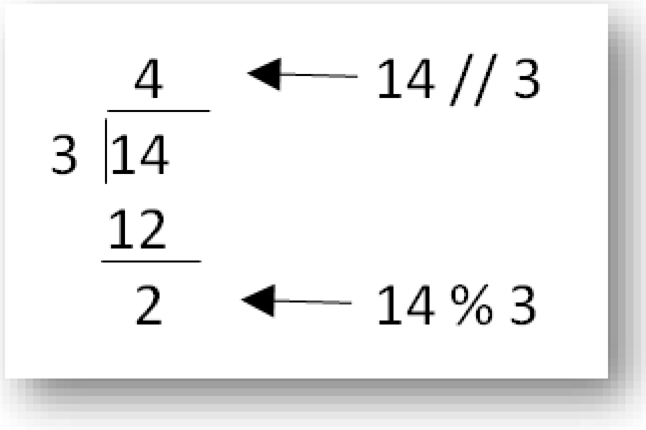


```
>>> 3*(2+3)  #Operations in the () go first
15
>>> 3*2+3
9

>>> 3*4**2 # Exponentiation ** is evaluated first.
48
>>> (3*4)**2
144
```

Two Other Integer Operators

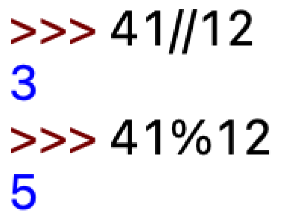
- Integer division operator
  - Written //
- Modulus operator
  - Written %

```
>>> 14//3   # Double division will return the quotient (the integer part)
4
>>> 14%3  # The % sign will return the remainder (modulo operation)
2
```



Example on // and %

- Convert 41 inches to feet & inches.
  (Rule of thumb: 1 foot = 12 inches.)



```
>>> 41//12
3
>>> 41%12
5
```

- Hence, 41 inches = 3 feet and 5 inches.

The abs Function

```
>>> help(abs)
Help on built-in function abs in module builtins:

abs(x, /)
    Return the absolute value of the argument.
```

```
>>> abs(23)
23
>>> abs(0)
0
>>> abs(-23)
23
```

Formula >

$$|x| = \begin{cases} x & \text{if } x \geq 0 \\ -x & \text{if } x < 0 \end{cases}$$

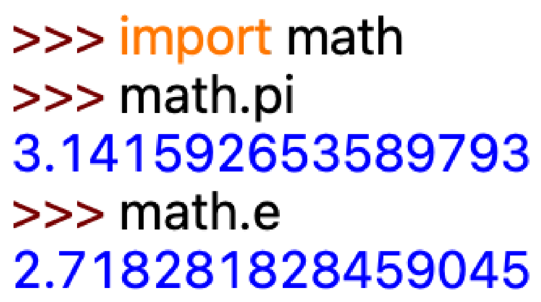
The `round` Function

```
>>> round(5.49999)
5
>>> round(5.5)
6


>>> round(5.14999, 1)
5.1
>>> round(5.15, 1)
5.2
```

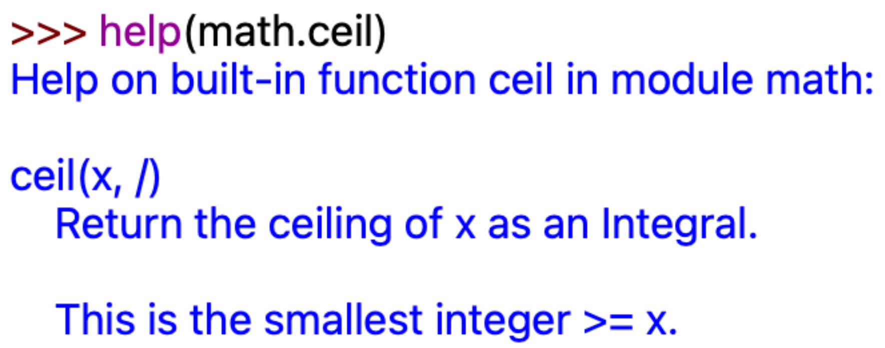
```
>>> help(round)
Help on built-in function round in module builtins:

round(number, ndigits=None)
    Round a number to a given precision in decimal digits.

    The return value is an integer if ndigits is omitted or None.  Otherwise
    the return value has the same type as the number.  ndigits may be negative.
```

The math module

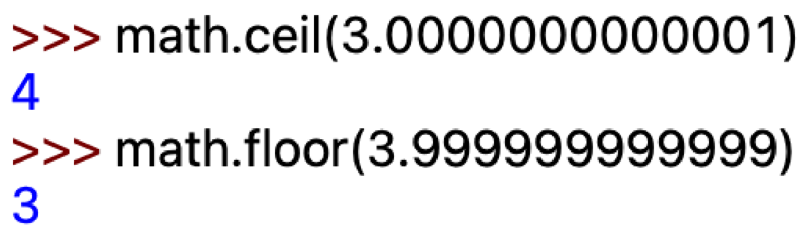
- The math module is a standard module in Python.
  - access to math functions and numbers



```
>>> import math
>>> math.pi
3.141592653589793
>>> math.e
2.718281828459045
```



```
>>> help(math.ceil)
Help on built-in function ceil in module math:

ceil(x, /)
    Return the ceiling of x as an Integral.

    This is the smallest integer >= x.
```



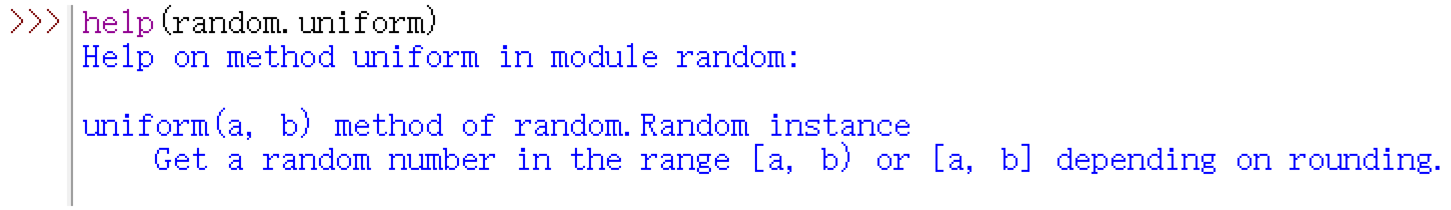
```
>>> math.ceil(3.0000000000001)
4
>>> math.floor(3.9999999999999)
3
```

The random module

- The random module can help generate random numbers.

```
>>> import random
>>> random.randint(1,100)
56
>>> random.randint(1,100)
45
>>> random.randint(1,100)
19
>>> random.randint(1,100)
28
>>> random.randint(1,100)
13
>>> random.randint(1,100)
69
```

```
>>> help(random.randint)
Help on method randint in module random:

randint(a, b) method of random.Random instance
    Return random integer in range [a, b], including both end points.
```

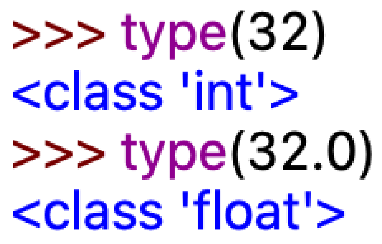


```
>>> help(random.uniform)
Help on method uniform in module random:

uniform(a, b) method of random.Random instance
    Get a random number in the range [a, b) or [a, b] depending on rounding.
```

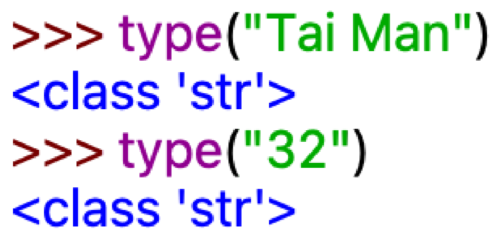
Object Types

Object types in Python

- Integer, float
- String
- List, tuple
- ...



```
>>> type(32)
<class 'int'>
>>> type(32.0)
<class 'float'>
```

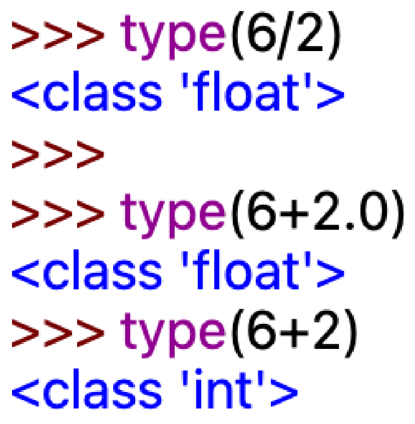


```
>>> type("Tai Man")
<class 'str'>
>>> type("32")
<class 'str'>
```

Object types after arithmetic operations

- Result of a division is always a float.

- The result of any other arithmetic operation is a float as long as at least one of the numbers is a float.



```
>>> type(6/2)
<class 'float'>
>>>
>>> type(6+2.0)
<class 'float'>
>>> type(6+2)
<class 'int'>
```

The `int` Function

```
>>> int(2.7)
2
>>> int(-2.7)
-2
>>> int(2)
2
```

```
>>> # The int function is different from math.floor for negative numbers.
>>> import math
>>> math.floor(-2.7)
-3
>>> int(-2.7)
-2
```

The **int** function leaves integers unchanged, and converts floating-point numbers to integers by discarding their decimal part.

The `int` Function

```
>>> # The int function can convert certain strings to integers.
>>> type("3")
<class 'str'>
>>> x=int("3")
>>> print(x)
3
>>> type(x)
<class 'int'>
```

```
>>> int("3.0")
Traceback (most recent call last):
  File "<pyshell#152>", line 1, in <module>
    int("3.0")
ValueError: invalid literal for int() with base 10: '3.0'
```

The `float` Function

```
>>> float(3) #The float function can convert integers to floats.
3.0
>>> float("3") #The float function can also convert certain strings to floats.
3.0
>>> float("3.1")
3.1
```

```
>>> float("4/2")
Traceback (most recent call last):
  File "<pyshell#189>", line 1, in <module>
    float("4/2")
ValueError: could not convert string to float: '4/2'
```

The `eval` Function

```
>>> eval("3")
3
>>> eval("3.2")
3.2
```

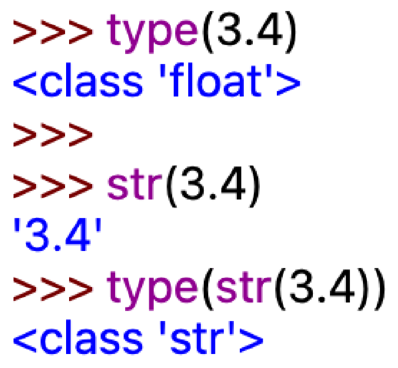
```
>>> eval("2*3")
6
>>> eval("4/2")
2.0
>>> eval("math.pi")
3.141592653589793
>>> eval("math.ceil(2.001)")
3
```

```
>>> x=3
>>> eval("x")
3
```

```
>>> eval(3)
Traceback (most recent call last):
  File "<pyshell#229>", line 1, in <module>
    eval(3)
TypeError: eval() arg 1 must be a string, bytes or code object
>>>
>>> eval("CityU")
Traceback (most recent call last):
  File "<pyshell#231>", line 1, in <module>
    eval("CityU")
  File "<string>", line 1, in <module>
NameError: name 'CityU' is not defined
```

The str Function

The **str** function can convert other objects into strings.



```
>>> type(3.4)
<class 'float'>
>>>
>>> str(3.4)
'3.4'
>>> type(str(3.4))
<class 'str'>
```

The input Function revisited

The **input** function will always convert user inputs into strings.

```
>>> num = input("Input a number: ")
Input a number: 3.2
>>> print(num)
3.2
>>> type(num) #Note that it is a string
<class 'str'>

>>> #One can use the eval function to convert the input into numeric literals.
>>> num = eval(input("Input a number: "))
Input a number: 3.2
>>> print(num)
3.2
>>> type(num)
<class 'float'>
```

The `print` Function revisited

- The `print` function displays items on the screen.

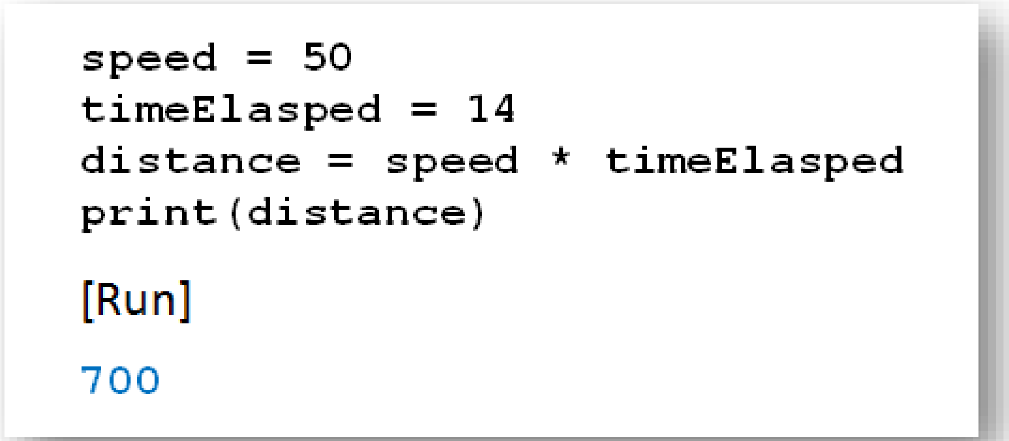
```
>>> print("CityU")   #Display a string
CityU
>>> print(1984)   #Display an integer
1984

>>> print("CityU", 1984)   # The print function can display several objects of different types
CityU 1984


>>> print(2+4)   # The print function can display evaluated expressions (e.g. numeric addition)
6
>>> print("CityU"+" HK")   # The print function can display evaluated expressions (e.g. string addition)
CityU HK

>>> print("41 inches =", 41//12, "feet and", 41%12, "inches.")  # Notice that 41 = 3*12 + 5.
41 inches = 3 feet and 5 inches.
```

Variables

Variables

- In mathematics problems, quantities are referred to by names
- Names given to the values are called **variables**

A Program that uses speed & time to calculate distance



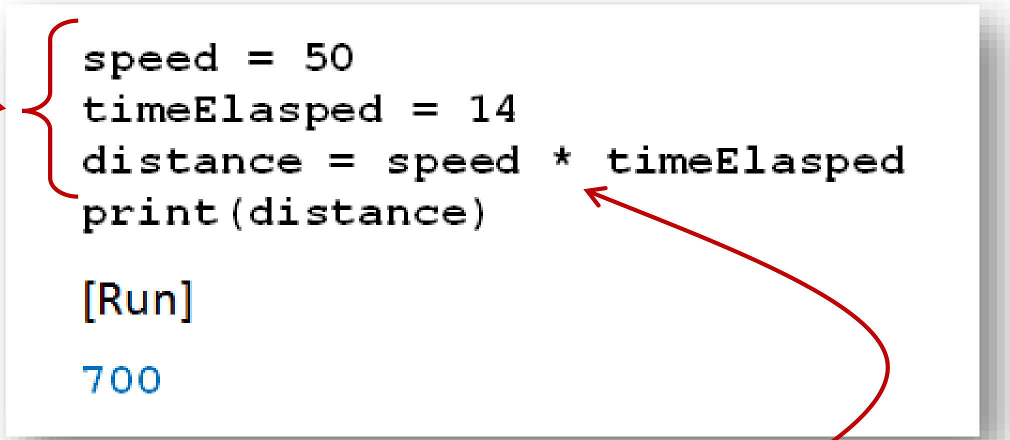
```
speed = 50
timeElasped = 14
distance = speed * timeElasped
print(distance)
```

[Run]

```
700
```

Variables

- Assignment statements



```
speed = 50
timeElasped = 14
distance = speed * timeElasped
print(distance)

[Run]

700
```

- Expression on right evaluated
  - That value assigned to variable

Variables

- Variable names in Python
  - Begin with letter or underscore _
  - Can only consist of letters, numbers, underscores

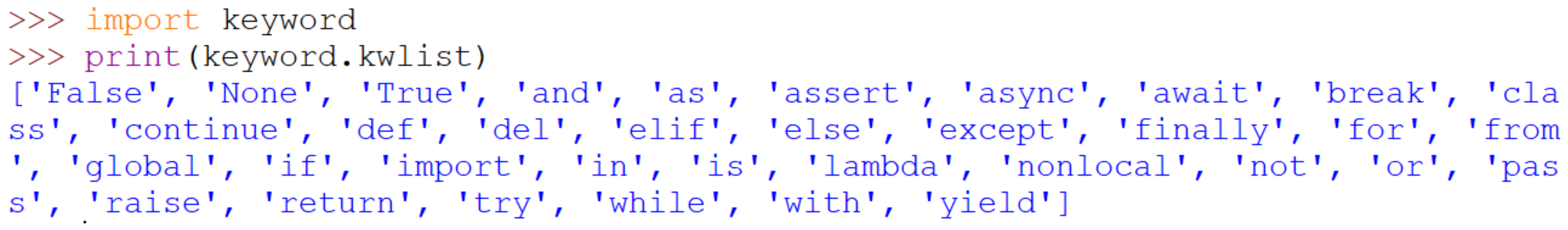
- Recommend using descriptive variable names

- Convention will be
  - Begin with lowercase
  - Use cap for additional “word” in the name
  - Example: **`rateOfChange`**

Variables

- Variable names in Python are case-sensitive

- There are thirty-five words, called **reserved words (**or **keywords)**
    - Have special meanings in Python (e.g. True, False)
    - Cannot be used as variable names
    - See Appendix B in the textbook
    - Or



```
>>> import keyword
>>> print(keyword.kwlist)
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'cla
ss', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from
', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pas
s', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

Augmented Assignments

- Remember: expression on the right side of assignment statement evaluated *before* assignment is made
  `var = var + 1`

- Python has special operator to accomplish the same thing
  `var += 1`

Augmented Assignments

```
>>> num = 5
>>> num += 1   # This is the same as   num = num + 1
>>> print(num)
6
```

```
>>> num = 5
>>> num -= 1  # This is the same as   num = num - 1
>>> print(num)
4
```

```
>>> num = 5
>>> num **= 2   # This is the same as   num = num ** 2
>>> print(num)
25
```

```
>>> num = 5
>>> num *= 2   # This is the same as   num = num * 2
>>> print(num)
10
```

```
>>> num = 5
>>> num /=4   # This is the same as   num = num / 4
>>> print(num)
1.25
```

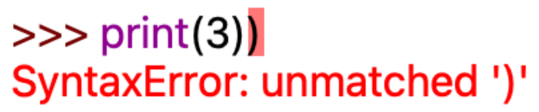
Error Types

Three types of errors

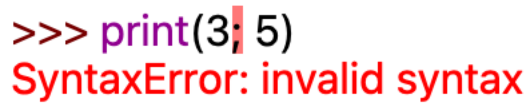
- Syntax errors
- Runtime errors
- Logic errors

Syntax Errors

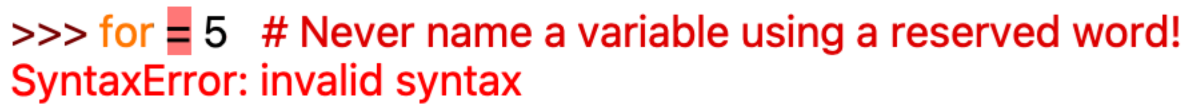
- Grammatical and punctuation errors are called *syntax errors*.



```
>>> print(3))
SyntaxError: unmatched ')'
```



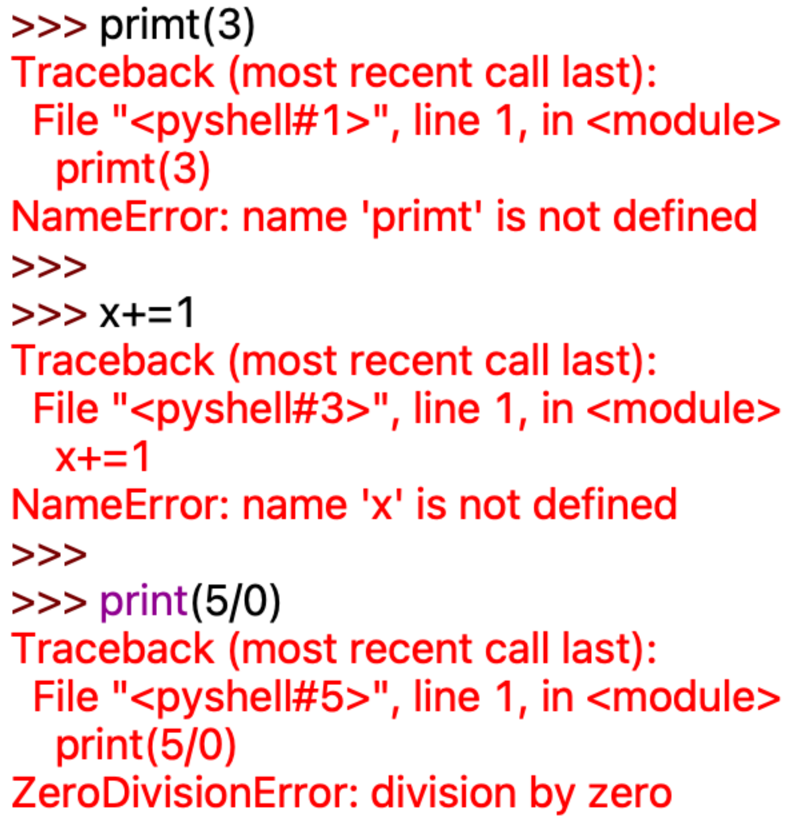
```
>>> print(3; 5)
SyntaxError: invalid syntax
```



```
>>> for = 5   # Never name a variable using a reserved word!
SyntaxError: invalid syntax
```

Runtime Errors

- Errors discovered while program is running called runtime errors or exceptions



```
>>> primt(3)
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    primt(3)
NameError: name 'primt' is not defined
>>>
>>> x+=1
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    x+=1
NameError: name 'x' is not defined
>>>
>>> print(5/0)
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    print(5/0)
ZeroDivisionError: division by zero
```

Logic Error

- Occurs when a program does not perform the way it was intended

- Example
  ```
  average = firstNum + secondNum / 2
  ```
  - Syntax correct, logic wrong: should be
    ```
    average = (firstNum + secondNum) / 2
    ```

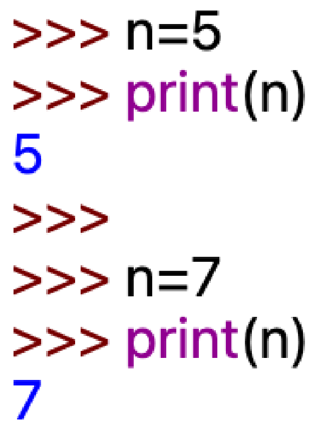
- Logic errors are the most difficult type of error to locate

Numeric Objects in Memory

- The variable *n* is said to **reference** (or **point to**) the number 5 in the memory location.

- When the second line of code is executed, Python sets aside a new memory location to hold the number 7 and redirects the variable *n* to point to the new memory location.

- The number 5 in memory is said to be *orphaned* or *abandoned*. Python will eventually remove the orphaned number from memory with a process called *garbage collection*.

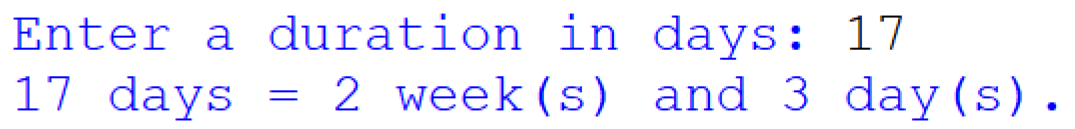


```
>>> n=5
>>> print(n)
5
>>>
>>> n=7
>>> print(n)
7
```

Lab 2

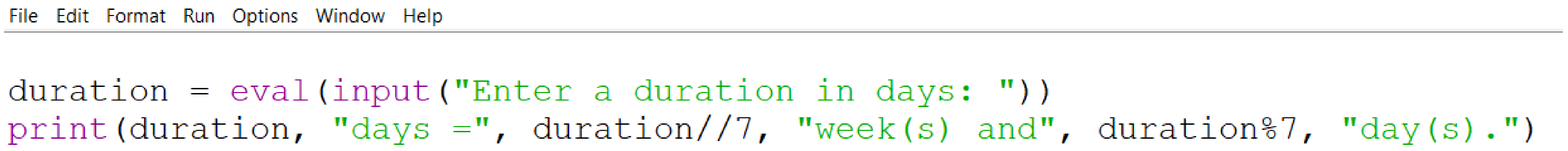
Example A: Days → Weeks & Days

- Let the user enter the number of days.
- Convert the input to # of weeks and # of days.



```
Enter a duration in days: 17
17 days = 2 week(s) and 3 day(s).
```

Example A: Days → Weeks & Days



```python
duration = eval(input("Enter a duration in days: "))
print(duration, "days =", duration//7, "week(s) and", duration%7, "day(s).")
```

Example B: Future value

- Write a program to calculate the future value of a savings account. Let the user set the initial amount, the number of years, and the annual interest rate. Assume the interest is compounded *annually*.
    - Present value is the sum of money that must be invested in order to achieve a specific future goal.
    - Future value is the dollar amount that will accrue over time when that sum is invested.
    - The present value is the amount you must invest in order to realize the future value.

```
Enter the initial amount: $1000
Enter the number of years: 10
Enter the annual interest rate (without the %): 0.01
The account balance in 10 years is $1104.62.
```

Example B: Future value

- Let PV be the initial amount, r be the annual rate, then the future amount after n years is

$$\mathrm{FV} = \mathrm{PV}(1 + r)^n$$

```
a = eval(input("Enter the initial amount: $"))
n = eval(input("Enter the number of years: "))
r = eval(input("Enter the annual interest rate (without the %): "))
futureValue = round(a*(1+r)**n, 2)

print("The account balance in", n, "years is $" + str(futureValue)+".")
```

- You may also use a loop.

Example C: Present value

- Your employer will award you a lump sum loyalty bonus of 100,000 at the end of your fifth year's employment. Write a program to calculate the present value of the bonus. Let the user set the annual interest rate. Assume the interest is compounded *monthly*.

```
Enter the annual interest rate (without the %): 0.05
The present value of the loyalty bonus in 5 years is $77920.54.
```

Example C: Present value

$$\mathrm{PV} = \frac{\mathrm{FV}}{(1+r)^n}$$

```
r = eval(input("Enter the annual interest rate (without the %): "))/12 # Monthly rate = annual rate / 12
presentValue = round(100000/(1+r)**(5*12), 2)  # Compounded monthly: 5*12 months in total.

print("The present value of the loyalty bonus in 5 years is $" + str(presentValue)+".")
```

- You may also write a loop.

Classwork 2: Future value of annuity

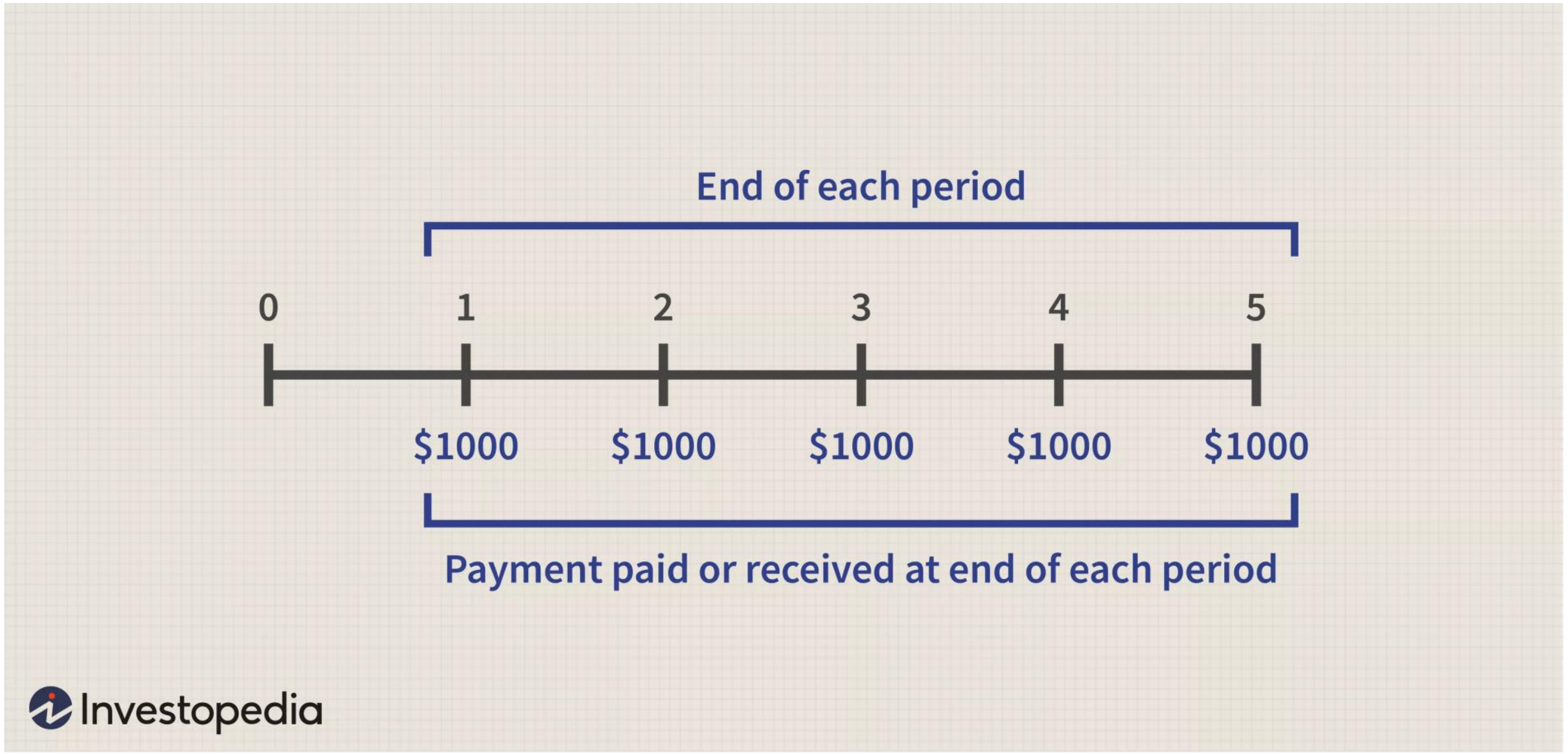
Assume you and your future employer each contribute HKD 1,500 to your Mandatory Provident Fund (MPF) account at the end of every month. Assume the initial account balance is 0, the return is compounded monthly, and the monthly rate of return is 1/12 of the annual rate. Write a Python program to calculate the future value of the MPF in 38 years.

1) Let the user set a target annual rate of return.
2) Round the answer to two decimal places for the amount in dollars, and one decimal place for the amount in millions.
3) The output should resemble the following.

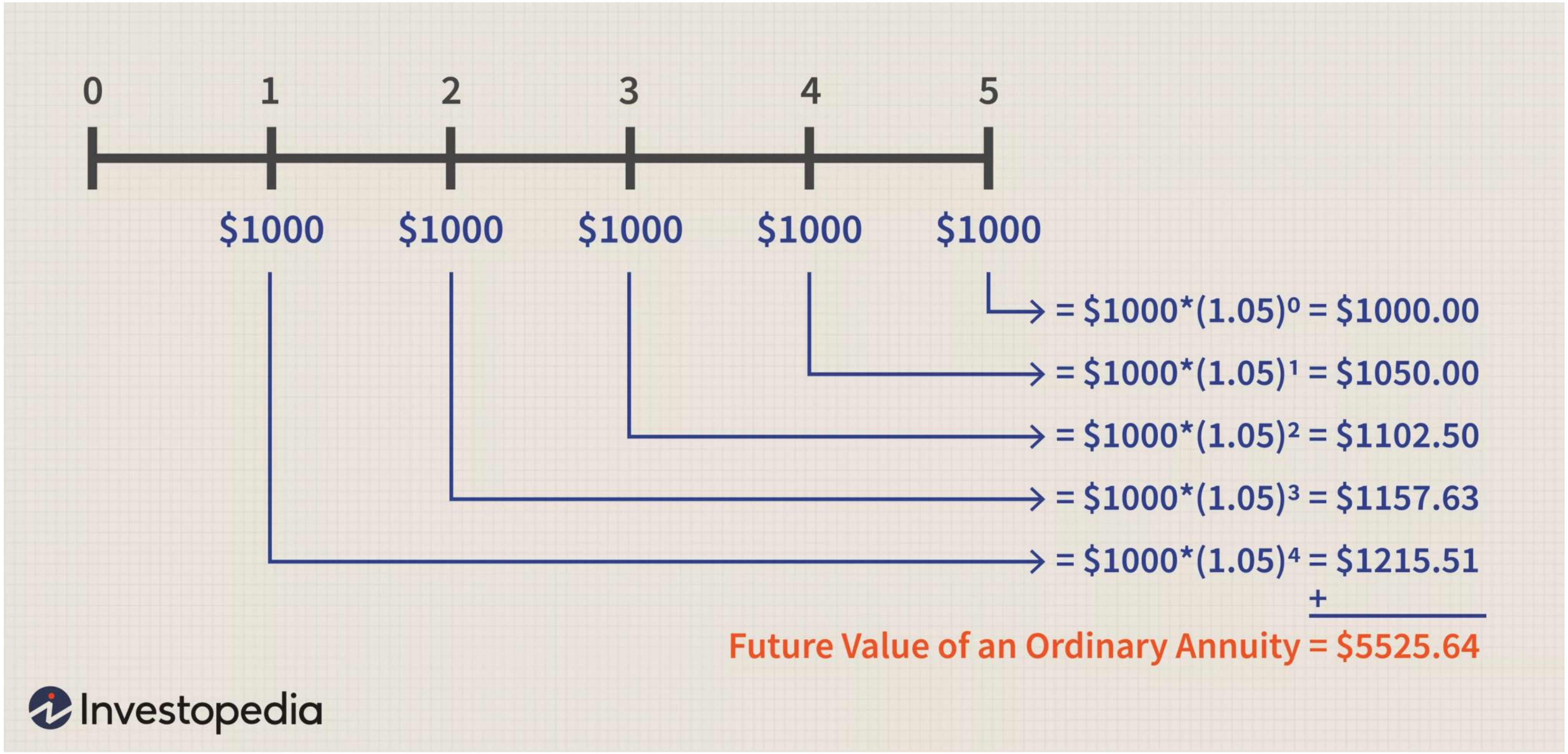
Upload the .py file to Canvas.

```
Enter your target ANNUAL rate of return (without the % sign): 0.03
With a total monthly contribution of $3000, your MPF will worth $2546787.95 after 38 years.
That is about 2.5 million(s).

Press ENTER to exit.
```

Classwork 2: Future value of annuity



Classwork 2: Future value of annuity



$= \$1000*(1.05)^0 = \$1000.00$

$= \$1000*(1.05)^1 = \$1050.00$

$= \$1000*(1.05)^2 = \$1102.50$

$= \$1000*(1.05)^3 = \$1157.63$

$= \$1000*(1.05)^4 = \$1215.51$

Future Value of an Ordinary Annuity = $5525.64

Classwork 2: Future value of annuity

- Assume the *month-end* contribution amount is c. The monthly rate is r, then the total value in n months is

$$\text{FV} = c + c(1+r) + c(1+r)^2 + \cdots + c(1+r)^{n-1} = \boldsymbol{c \cdot \frac{(1+r)^n - 1}{r}}$$

- You can also use loops.

Homework 2: Present value of annuity

- See the document on Canvas.
- Note that there is a cash flow every five years. Potential returns are compounded annually.