COMP7035

Python for Data Analytics and Artificial Intelligence

Renjie Wan, Jun Qi

3/9/2024

香港浸會大學
HONG KONG BAPTIST UNIVERSITY

DEPARTMENT OF
COMPUTER SCIENCE
計算機科學系

- **Lecturers**:

  Dr. Renjie WAN (Section I, Wed. 6:30 PM -- 9:20 PM)

  Email: renjiewan@comp.hkbu.edu.hk

  Phone: 3411 7633

  Office: Room 639, David C. Lam Building

  Dr. Jun QI (Section II, Tue. 2:30 PM – 5:20 PM)

  Email: jun-qi@comp.hkbu.edu.hk

  Phone: 3411 7705

  Office: Room 714, David C. Lam Building

- **Teaching Assistants:**

  Ziyuan LUO, ziyuanluo@life.hkbu.edu.hk

  Xiufeng HUANG, 23482974@life.hkbu.edu.hk

What we will learn?

I. Python Fundamentals
   A. Program control and logic
   B. Data types and structures
   C. Function
   D. File I/O

II. Numerical Computing and Data Visualization
Tools and libraries such as
   A. NumPy
   B. Matplotlib
   C. Seaborn

III. Exploratory Data Analysis (EDA) with Python
Tools and libraries such as
   A. Panda
   B. Sweetviz

IV. Artificial Intelligence and Machine Learning with Python
Tools and libraries such as
   A. Keras
   B. Scikit-learn

Logistics of our course

- Continuous assessments (40 %)
  - Two assignments
- One mid-term test (20%)
- Final examination (40%)

Two assignments are **computer-based**.
Two tests and the final examinations are all **paper-based.**

Logistics of our course

- Each course contains lecture and lab sessions totaling 170 minutes. The lecture takes about 1.5 hours, and the lab takes the remaining time.
- Lectures: **Learn the core knowledge**
- Lab session: **Do exercises together**
  - Exercises will be released in the lab session.
  - Please understand each exercise; they will be essential to our final examinations.

All announcements of this course will be released on Moodle.

Workload

- The only way to learn Python, is by writing Python a lot. So you are expected to put in effort.
- The examples used in our course will be provided for you. You can play them yourself.
- If you are new to programming, consider this a hard class where you will have to figure out quite a bit on your own. However, if you have a solid background in Python or any another language, this class should be pretty easy.
- If you are an expert in Python, please help your classmates.

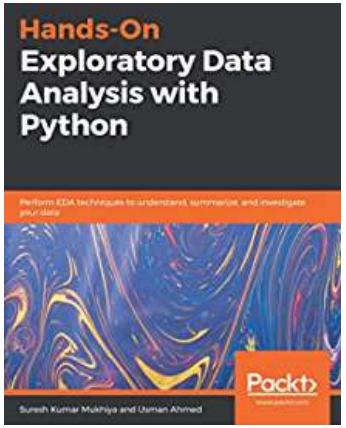
Suggestions for new beginners

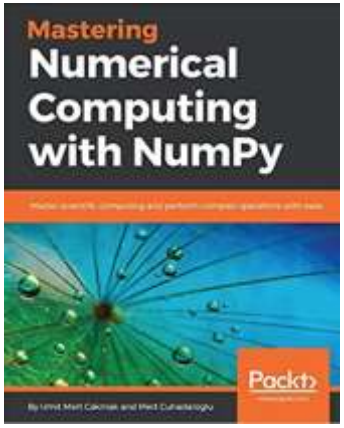
- The course is relatively short. We will go fast to cover many important knowledge about scientific python programming.
- Alternative: spend some time learning on your own (*Codecademy* / *Udacity etc*). There are so many excellent resources online these days.
- Try to learn Python from your friends.
- Try to learn Python from ChatGPT

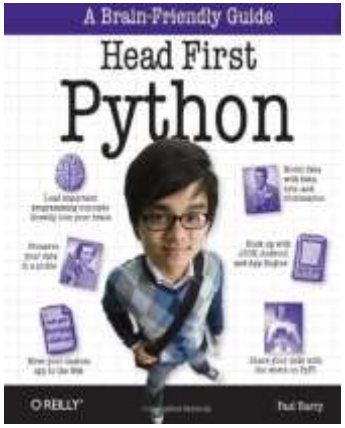
Our class would be very fast!!

References

- The internet is an excellent source, and Google is a perfect starting point.
- The official documentation is also good, always worth a try: https://docs.python.org/.







Chinese Resources: https://www.runoob.com/python/python-tutorial.html

References

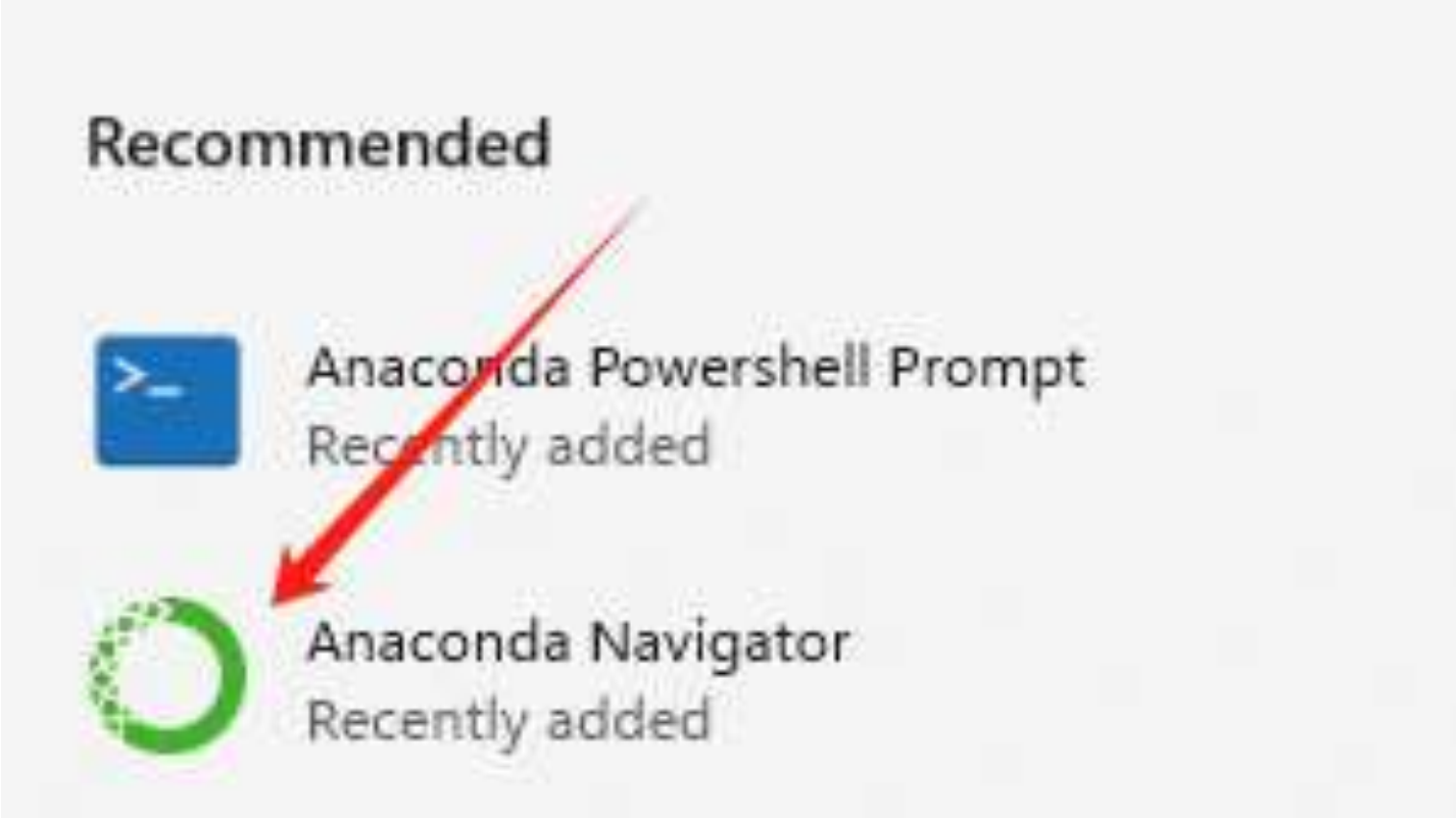
You can also ask **ChatGPT**!

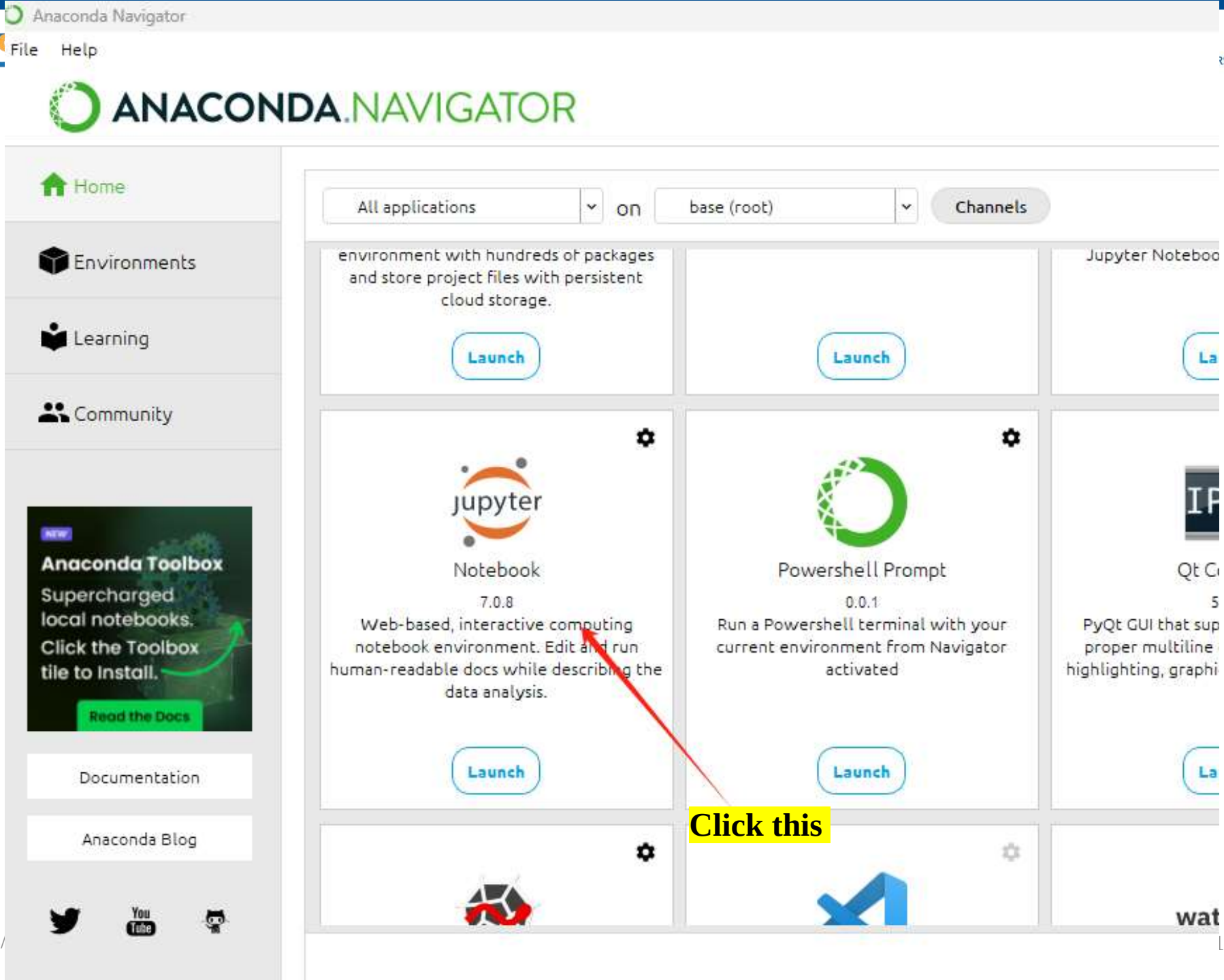
Try our university ChatGPT service

https://chatgpt.hkbu.edu.hk/

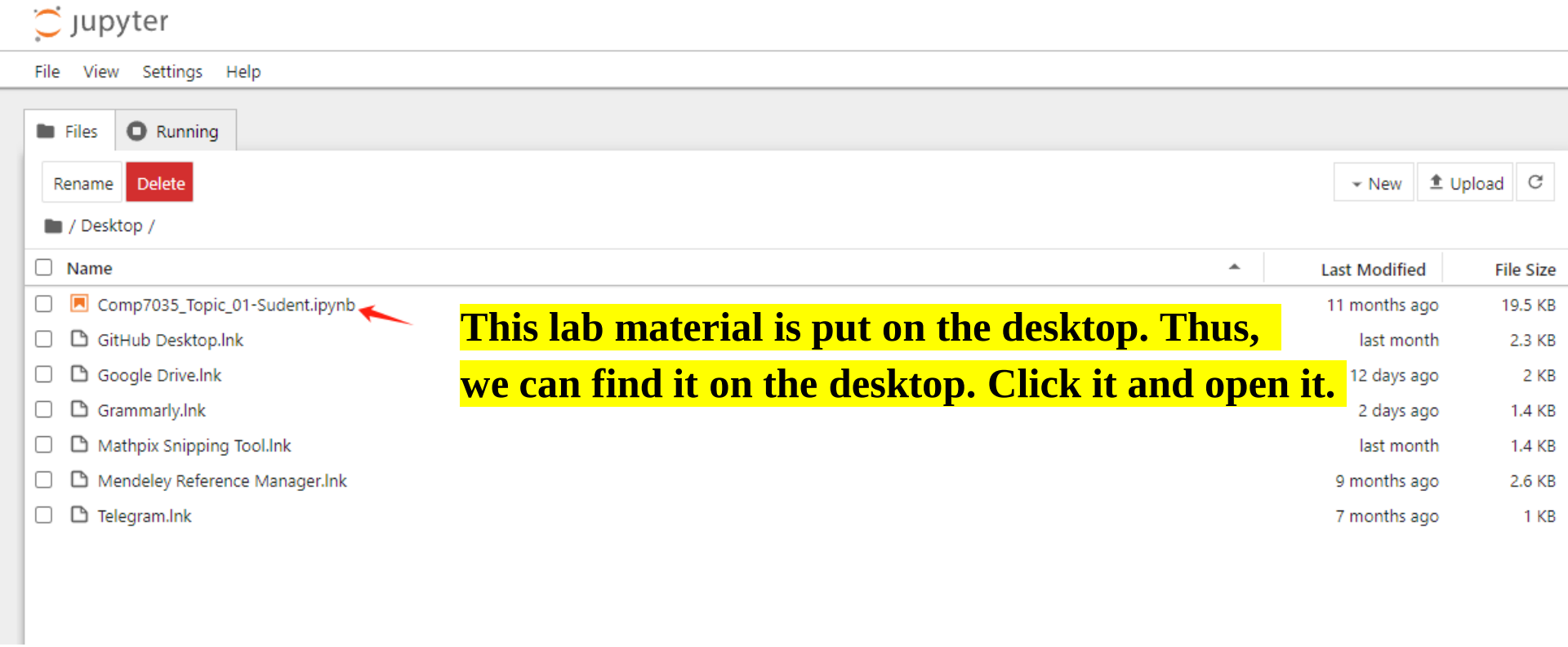
We use Anaconda in our class

Search Anaconda Navigator in your PC




Anaconda Navigator
File Help
ANACONDA.NAVIGATOR
Home
Environments
Learning
Community
Anaconda Toolbox
Supercharged local notebooks. Click the Toolbox tile to install.
Read the Docs
Documentation
Anaconda Blog
All applications
on
base (root)
Channels
environment with hundreds of packages and store project files with persistent cloud storage.
Launch
Launch
Jupyter Noteboo
Notebook
7.0.8
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.
Launch
Powershell Prompt
0.0.1
Run a Powershell terminal with your current environment from Navigator activated
Launch
PyQt GUI that sup
proper multiline
highlighting, graphi
Click this

Download your lab materials into your computer and open it via the following ways.



This lab material is put on the desktop. Thus, we can find it on the desktop. Click it and open it.

Lab Demo COMP7035 Week 1

Objective

- Use of the "print" command
- Use of the variable
- Use of different operators
- The basic of control statement

Click this button to see the result of your code.

Print a message

```
[ ]: print("How to pass this course?")
```

```
How to pass this course?
```

The code block

Try the two examples below

```
[ ]: import math

print(math.pi)
print(math.floor(1.5))
```

```
3.141592653589793
1
```

The RESULTS OF YOUR CODE

Arithmetic Operators:

```
[ ]: ## Print "40+20"
print("[Continuous Assessment]")
print("(Labs/Assignments + Practical Test(s))")
```

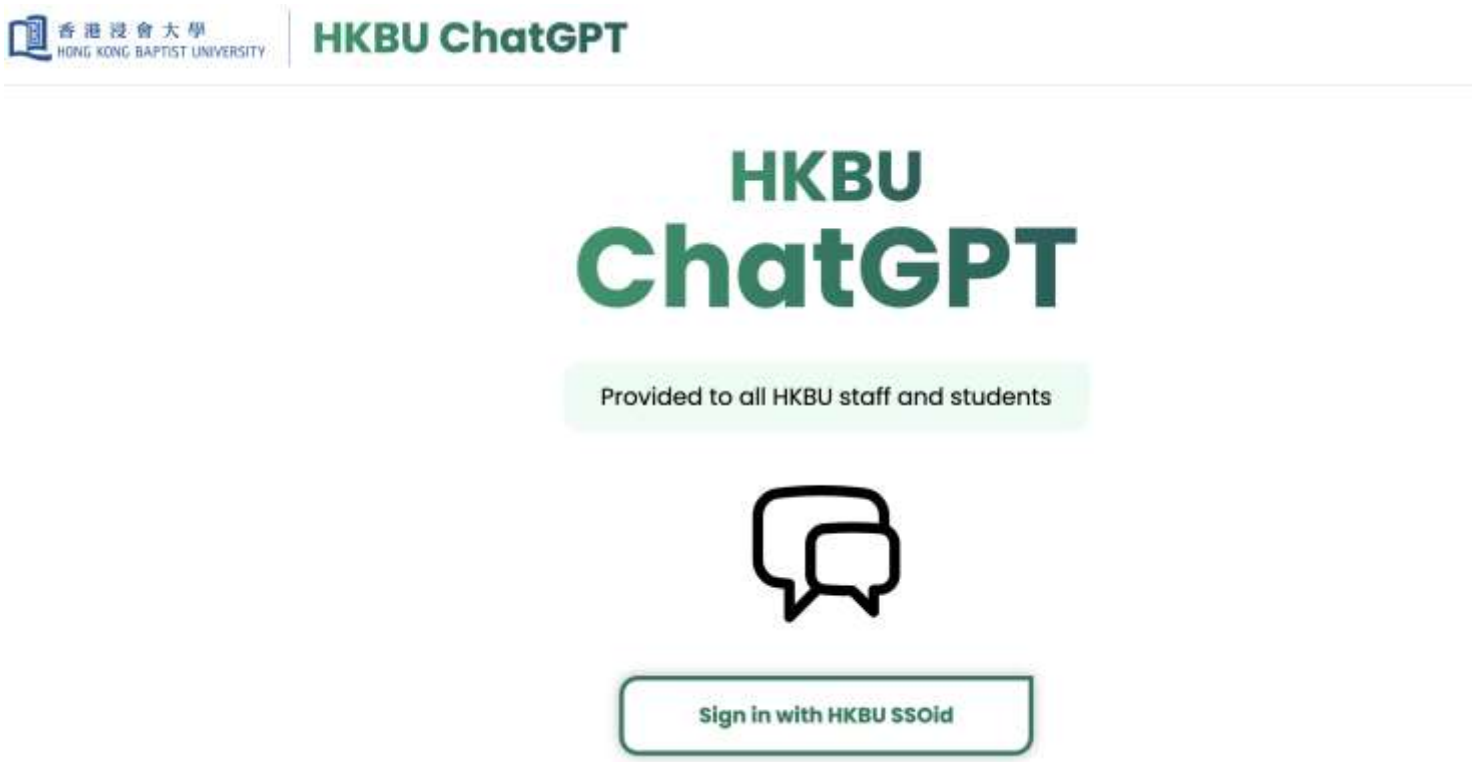
```
[ ]: ## Please Print 80 * 0.4 + 60 * 0.2
print("[Continuous Assessment]")
print("Your score: ")
```

addition

Let us begin!

Learn Python from ChatGPT

- Some samples from our university ChatGPT services



HKBU ChatGPT

Model : GPT-4 Turbo

What is the print in Python?

In Python, `print()` is a built-in function that is used to output data to the standard output device, which is usually the console. You can use it to display strings, numbers, and various other data types. The function converts the given object(s) into a string (if it is not already) and writes the result to the screen.

Here's a simple example of how to use `print()`:

```python
print("Hello, World!")
```

This will output:

```
Hello, World!
```

You can also print multiple items separated by commas, and `print()` will insert spaces between them by default:

```python
print("The answer is", 42)
```

Output:

Type your query

print statement

- We can print output to screen using the **print** command

```
print("Hello, world!")
```

```
print("Hello", "how are you?")
```

```
print("Hello", "how are you?", sep="---")
```

Values

- A value is the fundamental thing that a program manipulates.
- Values can be “Hello, world!”, 42, 12.34, True/False
- Values have types. . .

Types

- Each value has its corresponding types

  boolean True/False
  string “Hello, world!”
  integer 92
  float 3.1415

- Use **type** to find out the type of a value, as in

E5-E8

```
type(2/4)          float
type("Hello, World") String
type(True)         boolean
type(False)        boolean
```

Please try these codes to see the results

Variables

- One of the most basic and powerful concepts is that of a variable.
- A variable *assigns* a name to a value.

E9

```
x = 1
print(x)
```

```
#E11
y = "test"
print(y)
```

E10

```
import math
a = 1.5
print(math.floor(a))
```

```
#E12
x = 1
x = "string"
print(x)
```

Some hints about **variables**

- Almost always preferred to use variables over values:
  - Easier to update code
  - Easier to understand code (useful naming)

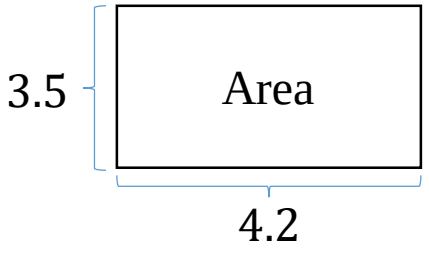
What does the following code do:

```
x = 4.2
y = 3.5
c = x * y
print(c)
```

Unclear about its usage

```
length = 4.2
height = 3.5
area = length * height
print(area)
```

It is calculating the area of a rectangle



Variables

- Variables are case-sensitive
- This will create two variables:

```
a = 4
A = "Sally"
```

- Python allows you to assign values to multiple variables in one line:

```
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```

Please try these codes to see the results

Variables

Please try these codes to see the results

- You can assign same values to same variables in one line.

```
x = y = z = "Orange"
print(x)
print(y)
print(z)
```

- The Python print() function is often used to output variables.

```
x = "Python is awesome"
print(x)
```

- In the print() function, you output multiple variables, separated by a comma:

```
x = "Python"
y = "is"
z = "awesome"
print(x, y, z)
```

Arithmetic Operators

```
print("[Continuous Assessment]")
print("(Labs/Assignments + Practical Test(s))")
print(40 + 20)
```

Results:

[Continuous Assessment]
(Labs/Assignments + Practical Test(s))
60

Python Arithmetic Operators

Operator	Description	Syntax
+	Addition: adds two operands	x + y
–	Subtraction: subtracts two operands	x – y
*	Multiplication: multiplies two operands	x * y
/	Division (float): divides the first operand by the second	x / y
//	Division (floor): divides the first operand by the second	x // y
%	Modulus: returns the remainder when the first operand is divided by the second	x % y
**	Power (Exponent): Returns first raised to power second	x ** y

Python Arithmetic Operators

- **Addition (+)**
- In Python, + is the addition operator. It is used to add 2 values.

```
val1 = 2
val2 = 3

# using the addition operator
res = val1 + val2
print(res)
```

Please try these codes to see the results

Python Arithmetic Operators

- **Subtraction Operator (-)**
- In Python, – is the subtraction operator. It is used to subtract the second value from the first value.

```
val1 = 2
val2 = 3

# using the subtraction operator
res = val1 - val2
print(res)
```

Please try these codes to see the results

Python Arithmetic Operators

- **Division Operator**
- In Python, / is the division operator. It is used to find the quotient when the first operand is divided by the second.

```
val1 = 3
val2 = 2

# using the division operator
res = val1 / val2
print(res)
```

Please try these codes to see the results

Python Arithmetic Operators

- **Modulus Operator**
- In Python, **%** is the modulus operator. It is used to find the remainder when the first operand is divided by the second.

```
val1 = 3
val2 = 2

# using the modulus operator
res = val1 % val2
print(res)
```

Please try these codes to see the results

Python Arithmetic Operators

- **Exponentiation Operator**
- In Python, ** is the exponentiation operator. It is used to raise the first operand to the power of the second.

```
val1 = 2
val2 = 3

# using the exponentiation operator
res = val1 ** val2
print(res)
```

Please try these codes to see the results

Python Arithmetic Operators

- **Floor Division Operator**
- In Python, // is used to conduct the floor division. It is used to find the floor of the quotient when the first operand is divided by the second.

```
val1 = 3
val2 = 2

# using the floor division
res = val1 // val2
print(res)
```

Please try these codes to see the results

Keywords

- Some Keywords cannot be used as as the variable names, they define structure and rules of a language.
- Python has 29 keywords, they include:
  - and
  - def
  - for
  - return
  - is
  - in
  - class

Introduction to different variables

String

- Strings in Python are surrounded by either single quotation marks or double quotation marks.
- 'hello' is the same as "hello".

Built-in methods

- Built-in methods are the predefined functions in Python that allow you to use the basic properties of strings and numbers in your programs.
- For Strings, several built-in methods are provided for different purposes.

Methods of Strings

- Python has a set of built-in methods that you can use on strings.
- **capitalize(): Converts the first character to upper case**

```
txt = "hello, and welcome to my world."

x = txt.capitalize()

print(x)
```

Methods of Strings

- Python has a set of built-in methods that you can use on strings.
- **capitalize(): Converts the first character to upper case**

```
txt = "hello, and welcome to my world."

x = txt.capitalize()

print(x)
```

Hello, and welcome to my world.

Methods of Strings

- Python has a set of built-in methods that you can use on strings.
- **count(): Returns the number of times a specified value occurs in a string**

```
txt = "I love apples, apple are my favorite fruit"

x = txt.count("apple")

print(x)
```

Methods of Strings

- Python has a set of built-in methods that you can use on strings.
- **count(): Returns the number of times a specified value occurs in a string**

```
txt = "I love apples, apple are my favorite fruit"

x = txt.count("apple")

print(x)
```

The result you can see: 2

Methods of Strings

- Python has a set of built-in methods that you can use on strings.
- **isdigit(): Returns True if all characters in the string are digits**

```
txt = "50800"

x = txt.isdigit()

print(x)
```

Methods of Strings

- Python has a set of built-in methods that you can use on strings.
- **isdigit(): Returns True if all characters in the string are digits**

```
txt = "50800"

x = txt.isdigit()

print(x)
```

The result you can see: True

Methods of Strings

- Python has a set of built-in methods that you can use on strings.
- **Replace(): Returns a string where a specified value is replaced with a specified value**

```
txt = "I like bananas"

x = txt.replace("bananas", "apples")

print(x)
```

Methods of Strings

- Python has a set of built-in methods that you can use on strings.
- **Replace(): Returns a string where a specified value is replaced with a specified value**

```
txt = "I like bananas"

x = txt.replace("bananas", "apples")

print(x)
```

The result you can see: I like apples

String Concatenation

- To concatenate, or combine, two strings you can use the + operator.
- Merge variable a with variable b into variable c:

```
a = "Hello"
b = "World"
c = a + b
print(c)
```

The result you can see: HelloWorld

String Concatenation

- To concatenate, or combine, two strings you can use the + operator.
- Merge variable a with variable b into variable c:

```
a = "Hello"
b = "World"
c = a + b
print(c)
```

HelloWorld

How can we change it to "Hello World"?

Python Numbers

```
x = 1     # int
y = 2.8   # float
```

- Two common numerical types in Python: int and float
- To verify their types, you can use type()
- Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.

```
x = 1
y = 35656222554887711
z = -3255522

print(type(x))
print(type(y))
print(type(z))
```

Python Numbers

- Float, or "floating point number" is a number, positive or negative, containing one or more decimals.

```
x = 1.10
y = 1.0
z = -35.59

print(type(x))
print(type(y))
print(type(z))
```

```
x = 35e3
y = 12E4
z = -87.7e100

print(type(x))
print(type(y))
print(type(z))
```

- Float can also be scientific numbers with an "e" to indicate the power of 10.

Booleans

- In programming, you often need to know if an expression is True or False.
- You can evaluate any expression in Python and get one of two answers: True or False.
- When you compare two values, the expression is evaluated, and Python returns the Boolean answer:
-

```
print(10 > 9)
print(10 == 9)
print(10 < 9)
```

Booleans

- In programming, you often need to know if an expression is True or False.
- You can evaluate any expression in Python and get one of two answers: True or False.
- When you compare two values, the expression is evaluated, and Python returns the Boolean answer:
-

```
print(10 > 9)
True
print(10 == 9)
False
print(10 < 9)
False
```

Booleans

- Almost any value is evaluated as True if it has some content.
- Any string is True, except empty strings.
- Any number is True except 0.
- Any list, tuple, set, and dictionary are True, except empty ones.

```
bool("abc")
bool(123)
bool(["apple", "cherry", "banana"])
```

Booleans

- There are not many values that evaluate to False, except empty values, such as (), [], {}, "", the number 0, and the value None. And, of course, the value False evaluates to False.

```
bool(False)
bool(None)
bool(0)
bool("")
bool(())
bool([])
bool({})
```

Booleans

- Boolean expressions:
  - == equals: 5==5 yields True
  - != does not equal: 1!= 1 yields False, while 2!=1 yields True
  - > greater: 2 > 1 yields True, while 1 > 2 yields False
  - >= greater than or equal: 5 >= 5 yields True
  - Similarly, we have < and <=

```
[ ]  #E22
     2 < 1
```
False

```
[ ]  #E23
     2 == 2
```
True

```
[ ]  #E24
     2 != 2
```
False

Python Conditions

- Python supports the usual logical conditions from mathematics:
  - Equals: a == b
  - Not Equals: a != b
  - Less than a < b
  - Less than or equal to a <= b
  - Greater than: a > b
  - Greater than or equal to a >= b
- These conditions can be used in several ways, commonly in "if statements" and loops.
- An "if statement" is written by using the if keyword.

Control statement

- Control statements allow you to do more complicated tasks:
  - for
  - while
  - if

if

- Using **if**, we can execute part of a program conditional on some true statement.

```
#E41
if True:
    Conduct statement
```

True: 1==1, 2>1, 3!=1, 10>0, 0<2

False: 1!=1, 2<1, 3==1, 10<0, 0>2

`If 2` is greater than `0`, then `print "2 is greater than 0".`

if

- Using **if**, we can execute part of a program conditional on some true statement.

```
#E41
if True:
    Conduct statement
```

True: 1==1, 2>1, 3!=1, 10>0, 0<2

False: 1!=1, 2<1, 3==1, 10<0, 0>2

If 2 is greater than 0, then `print "2 is greater than 0".`

```
#E41
if 2 > 0:
    print("Two is greater than zero")
```

```
Two is greater than zero
```

if

- Short hand if: If you have only one statement to execute, you can put it on the same line as the if statement.

```
if 2 > 0:
    print("Two is greater than zero")
```

```
if 2 > 0: print("Two is greater than zero")
```

If a is greater than 0 and b is greater than 0, then the two numbers are greater than 0

How can we express "**and**" in this place?

Program **logic**

- **and**: Logic **AND**: True if both the operands are true.
  - True and False yields False
- **or**: Logic **OR**: True if either of the operands is true.
  - True or False yields True
- **not**: Logic NOT: True if operand is false.
  - not True yields False, not False yields True

```
[2]  #E17
     True and False

     False
```

```
[6]  #E19
     not True

     False
```

```
[7]  #E18
     True or False

     True
```

```
[5]  #E20
     not False

     True
```

Use if with logic **and**

- Think about a scenario:
  - You have two numbers and want to know whether they are all greater than 0

If a is greater than 0 and b is greater than 0, then the two numbers are greater than 0
(a > 0 and b > 0)

```
#E43
a = 10
b = 10
#Write your code here
```

Result: The numbers are greater than 0

Use if with logic **and**

- Think about a scenario:
  - You have two numbers and want to know whether they are all greater than 0

If a is greater than 0 and b is greater than 0, then the two numbers are greater than 0
(a > 0 and b > 0)

```
#E43
a = 10
b = 10
if a > 0 and b > 0:
  print("The numbers are greater than 0")
```

Result: The numbers are greater than 0

Exercise - and

- Test if a is greater than b, AND if c is greater than a:
- "and" means the two things happen simultaneously.

```
a = 200
b = 33
c = 500
if a > b and c > a:
  print("Both conditions are True")
```

Use if with logic **or**

- Think about a scenario:
  - Today is Saturday; you want to know whether you need to go to work on the weekend?

`if` today is on Saturday `or` Sunday.
On the two days, you just need to rest at home

```
#E43
today = 'Sunday'
if today=='Sunday' or today=='Saturday':
  print('Today is off. Rest at home.')
```

if-else statement

- We can add more conditions to the **if** statement using **else**
- **else** is used to cover conditions not covered by the **if**

if x is equal to y, then display “The two number are equal”

```
x = 1
y = 1
if x == y:
    print("The two numbers are equal")
```

if-else statement

- We can add more conditions to the **if** statement using **else**
- **else** is used to cover conditions not covered by the **if**

```
if x is equal to y, then display “The two number are equal”
else:
display “The two numbers are not equal”.
```

```
#E42
x = 1
y = 2

#write your code here
if x == y:
    print("The two number are equal")
else:
    print("The two numbers are not equal")
```

How to express “otherwise” in this place?

if-else statement

- We can add more conditions to the **if** statement using **else**
- **else** is used to cover conditions not covered by the **if**

```
if x is equal to y, then display “The two number are equal”
else
display “The two numbers are not equal”.
```

Please try these codes to see the results

```
#E42
x = 1
y = 2

#write your code here
if x == y:
    print("The two number are equal")
else:
    print("The two numbers are not equal")
```

if-else statement

- We can add more conditions to the **if** statement using **else**
- **else** is used to cover conditions not covered by the **if**

Please try these codes to see the results

```
#E44
today = 'Friday'
if today=='Sunday' or today=='Saturday':
    print('Today is off. Rest at home')
else:
    print('go to work')
```

Indentation

- In Python, blocks of code are defined using indentation.
- This means that everything indented after an **if** statement is only executed if the statement is true.
- If the statement is **False**, the program skips all indented code and resumes at the first line of unindented code.

```
#E-Indentation-1
if 2<1:
  print("2<1")
  print("1>2")
```

```
#E-Indentation-2
if 2<1:
  print("2<1")
print("1>2")
```

Please try these codes to see the results

Indentation

- In Python, blocks of code are defined using indentation.
- This means that everything indented after an **if** statement is only executed if the statement is true.
- If the statement is **False**, the program skips all indented code and resumes at the first line of unindented code.

```
#E-Indentation-1
if 2<1:
  print("2<1")
  print("1>2")
```

show nothing

```
#E-Indentation-2
if 2<1:
  print("2<1")
print("1>2")
```

1>2

for loops

```
for item in iterable:
    statement(s)
```

- When some actions are repeated, this can be achieved by a **for** loop.

```
#E46
print("loop 1")
for i in range(5): # default - start at 0, increment by 1
    print(i)

print("\nloop 2")
for i in range(2, 10): # inputs are start, stop
    print(i)
```

Please try these codes to see the results

Here, **range(n)** gives us a **list** with integers 0,……, $n-1$.
For example, **range(5)** gives us a list with 0, 1, 2, 3, 4

for loops

```
for item in iterable:
    statement(s)
```

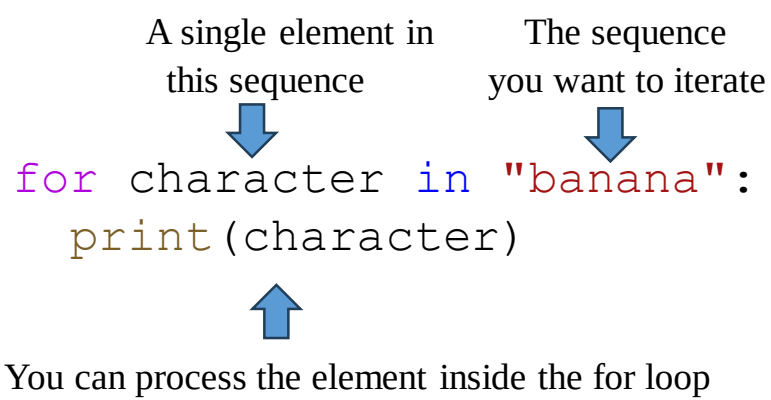
- When some actions are repeated, this can be achieved by a **for** loop.

```
#E46
print("loop 1")
for i in range(5): # default - start at 0, increment by 1
    print(i)
                                Result: 0 1 2 3 4
print("\nloop 2")
for i in range(2, 10): # inputs are start, stop
    print(i)
                                Result: 2 3 4 5 6 7 8 9
```

Here, **range(n)** gives us a **list** with integers 0,……, $n-1$.
For example, **range(5)** gives us a list with 0, 1, 2, 3, 4

for loops

- A for loop is used for iterating over a sequence (a list, a tuple, a dictionary, a set, or a string).
- Use string value as an example




```
for character in "banana":
    print(character)
```

```
b
a
n
a
n
a
```

More about range()

- The range() function returns a sequence of numbers, starting from 0 by default, incrementing by 1 (by default), and stopping before a specified number.
- Syntax: `range(start, stop, step)`

start	**Optional. An integer number specifying at which position to start. Default is 0**
stop	**Required. An integer number specifying at which position to stop (not included).**
step	**Optional. An integer number specifying the incrementation. Default is 1**

Question 1: Create a sequence of numbers from 3 to 5, and print each item.

How to write the **range** function?

If you want to print 5, you need to write the stop number as 6, since it is not included!

More about range()

- The range() function returns a sequence of numbers, starting from 0 by default, incrementing by 1 (by default), and stopping before a specified number.
- Syntax: `range(start, stop, step)`

start	**Optional. An integer number specifying at which position to start. Default is 0**
stop	**Required. An integer number specifying at which position to stop (not included).**
step	**Optional. An integer number specifying the incrementation. Default is 1**

Question 1: Create a sequence of numbers from 3 to 5, and print each item.

```
x = range(3, 6)
for n in x:
  print(n)
```

More about range()

- The range() function returns a sequence of numbers, starting from 0 by default, incrementing by 1 (by default), and stopping before a specified number.
- Syntax: `range(start, stop, step)`

start	**Optional. An integer number specifying at which position to start. Default is 0**
stop	**Required. An integer number specifying at which position to stop (not included).**
step	**Optional. An integer number specifying the incrementation. Default is 1**

Create a sequence of numbers from 3 to 19, but increment by 2 instead of 1:

```
x = range(3, 20, 2)
for n in x:
  print(n)
```

More about range()

- The range() function returns a sequence of numbers, starting from 0 by default, incrementing by 1 (by default), and stopping before a specified number.
- Syntax: `range(start, stop, step)`

start	**Optional. An integer number specifying at which position to start. Default is 0**
stop	**Required. An integer number specifying at which position to stop (not included).**
step	**Optional. An integer number specifying the incrementation. Default is 1**

Create a sequence of numbers from 3 to 19, but increment by 2 instead of 1:

while loops

- When we do not know how many iterations are needed, we can use **while**

```
#E47
i = 1
while i < 10:
    print(i)
    i = i + 1
```

Please try these codes to see the results

while *i* is smaller than 10, print its value and increase its value by 1

while loops

- When we do not know how many iterations are needed, we can use **while**

```
#E47
i = 1
while i < 10:
    print(i)
    i = i + 1
```

Result: 1 2 3 4 5 6 7 8 9

while i is smaller than 10, print its value and increase its value by 1

continue

- **continue** continues with the next iteration of the smallest enclosing loop.

```
#E48
for num in range(2, 10):
    if num % 2 == 0:
        continue # this jumps us back to the top
    print(f"Found {num}, an odd number")
```

break

- The **break** statement lets us jump out of the smallest enclosing **for** or **while** loop.

You have a word. for all letters in this word, find whether “e” exists or not. If it indeed exits, immediately stop the program

```
#E49
for letter in "Hello":
    if letter == "e":
        print(letter)
        break
```

pass

- The **pass** statement does nothing, which can be helpful when working on something and you want to implement some parts of your code later.

```
a = 10
b = 20
if(a<b):
  pass
else:
  print("b<a")
```

```
li =['a', 'b', 'c', 'd']
for i in li:
  if(i =='a'):
    pass
  else:
    print(i)
```

Exercise 1

- Judge whether the given year is a leap year (闰年).

If (year is divisible by 4 and not divisible by 100) or (year is divisible by 400) then
    Print year, "is a leap year."
 Else
    Print year, "is not a leap year."

Exercise 1

- Write a program that prints the largest of three given numbers.

- If num1 is greater than or equal to num2 and num1 is greater than or equal to num3, then the largest is assigned the value of num1.
- Otherwise, if num2 is greater than or equal to num1 and num2 is greater than or equal to num3, then the largest is assigned the value of num2.
- Else, the largest is assigned the value of num3.

Exercise 2

- See the lab materials

Exercise 3

- See the lab materials

Exercise 1

- Write a program that prints the largest of three given numbers.

```
num1 = 10
num2 = 14
num3 = 12

if (num1 >= num2) and (num1 >= num3):
   largest = num1
elif (num2 >= num1) and (num2 >= num3):
   largest = num2
else:
   largest = num3

print("The largest number is", largest)
```