W9: Intro to Python - Day 1

Fei-Man Hsu, PhD

UCLA QCBio Collaboratory, Fall 2025

This is the notebook for the first day of the Collaboratory Workshop, *Intro to Python*.

Outline

Day 1:

- Concept and setup
- Basic data types and variables
- If statements

Day 2:

- Loops
- More data types: Collections (lists, sets, dictionaries)

Day 3:

- Functions
- Reading/Writing Files
- Libraries

Assignments are mandatory for credits

Setup

If you want your notebook to run on Google's cloud

1. Open a web browser on your computer, and **log in to your Google account**.

2. Go to https://colab.research.google.com/

3. Upload the notebook (`.ipynb` file)

4. Open it.

If you want to run your local installation of Python

1. Install Anaconda.
2. Open Anaconda Navigator.
   - Mac: Open Spotlight Search (Command + Space bar) and search "Anaconda Navigator"
   - Windows: Look for Anaconda in your Start menu.
3. Launch Jupyter Notebook.

Once launched, the Jupyter Dashboard opens in a web browser.

- Files relative to your home directory is displayed.
- You can create folders and a new notebook.

In a notebook, you can write/edit "cells" of code or write a cell with a text, markdown-formatted.

- https://www.ibm.com/docs/en/watson-studio-local/1.2.3?topic=notebooks-markdown-jupyter-cheatsheet for some syntax.

You are running a **Jupyter Notebook** now.

- Open-source
- Create and share documents
- Contains live code and hypertext "literate programming"
- Similar applications
    - R: R Markdown Notebook
    - Matlab: Matlab Live Script

You can run code in a cell using the "Run" button or by pressing `Shift + Enter` on the keyboard.

This is **not** the only way to run Python code. You will learn other ways to run your code on Day 3.

Basic data types in Python include:

- Text Type: `str`
- Numeric Types: `int`, `float`, `complex`
- Sequence Types: `list`, `tuple`, `range`
- Mapping Type: `dict`
- Boolean Type: `bool`

We will deal with some of these basic data types.

Your First Program: "Hello, world!"

It is a common practice in many programming languagues to start with a program that prints out "Hello, world" to your screen. It shows basic syntax of the language, and you can check if the system is set up properly.

Type and run the following in the code cell below:

```
print("Hello, world!") # Print a string
```

In [2]:
```
print("Hello, world!")
```

```
Hello, world!
```

Strings (`str`)

Strings in python represent text information.

- String literals are surrounded by either a pair of single quotation marks (`'`) or a pair of double quotation marks (`"`).
- For each literal, only one type of quotation mark should start and end text string.
- `"hello"` is the same is `'hello'`.

In [3]:
```
"hello"
```

Out[3]:
```
'hello'
```

In [4]:
```
'hello'
```

Out[4]:
```
'hello'
```

In [8]:
```
"hello"
```

Out[8]:
```
'hello'
```

There are special characters that can be used in the string:

- `\n` - new line
- `\t` - tab
- If you need a quotation mark of the same type as the enclosing ones inside a string, it should be escaped using a backslash (`\`): `" \" "` or `' \' '`.
- You can read more at https://en.wikipedia.org/wiki/Escape_character.

Note: There are two types of cells in Jupyter Notebook: Code cell and Markdown (Text) cell.

- Code cells will have an heading with square bracket indicating cell execution order.
- Throughout this workshop, you will be asked to type something to a code cell and execute it.
- Sometimes, what to type in the code cell will be given directly. I encourage you to type in the code yourself and get your hand dirty -- that's more effective way to learn programming.

The `print(x)` **function** prints a string `x` to your screen.

Function is a a block of code which only runs when it is called. You can pass data, known as *parameters*, into a function. We will learn to how to use functions, and how to define functions later in this workshop.

Comments

Comments in Python start with a hash sign (`#`). Anything after a `#` will not be interpreted as a part of a program.

In [9]:
```python
print("Hello") # print hello
```
```
Hello
```

Multiline string enclosed in triple quotation marks(`"""` or `'''`) can also be used instead of a hash sign. Technically, this is a string literal, but Python interpreter will ignore it if it is not assigned to a variable.

In [11]:
```python
''' This is my first program '''
print("hello, world!")
```
```
hello, world!
```

Exercise (3 min)

Replace `<x>` and `<y>` with your own words and run the following code:

```python
print("<x>")
print('<x>')
print("<x> \t <y>")
print("<x> \n <y>")
print("<x> " <y>") # raises error
print("<x> \"<y>") # error fix
print('<x> ' <y>') # raises error
print('<x> \' <y>') # error fix
print("<x> ' <y>") # another error fix
print("<x> \\ <y>")
```

In [19]:
```python
print("Apple \" Banana")
```
```
Apple " Banana
```

Numbers

There are three types of numbers: integers (`int`), floating-point (`float`), and complex (`complex`) numbers.

Integers are for whole numbers.

```python
x = 1000
`
```

Floating-point numbers are for real values that are not integers:

```
y = 1.1
z = 5e-6
```

Arithmetic operations

- `+`, `-`
- `*` - multiplication
- `**` - exponentiation
- `/` - numeric division, the result is `float`
- `//` - integer division, the result is in whole number
- `%` - modulo operation
- `()` - priority

In [20]:
```
5 + 7
```
Out[20]:
```
12
```

In [21]:
```
(5 + 7) * 12
```
Out[21]:
```
144
```

In [22]:
```
3 ** 2
```
Out[22]:
```
9
```

Integer division (quotient) is a floor function of division. Floor function of x gives the largest integer less than or equal to x. *Integer division between `int` gives another integer.*

```
5 // 3
```

```
1
```

Modulo operation gives a remainder of division. Remainder is an integer.

```
5 % 3
```

```
2
```

Numeric division of two gives mathematical division, numeric division between integers does not result in an integer!

```
5 / 3
```

```
1.6666666666666667
```

This result is a `float`.

In [25]:
```
5 % 3
```
Out[25]:
```
2
```

Variables and Literals

- **Variables** reserves a space in computer memory to store data.
- Variables can store strings, numbers, lists, and many others types of data.
- Variables have their own name that allow access, and modify the data.
- **Literals** represent exact values that can be stored in variables.
- `=` assigns values to a variable.

```
 my_variable = "My first variable!"
(variable)     (literal)
```

A variable `my_variable` is defined, and is initialized with the string literal `"My first variable!"`. The equal sign means assignment of a value to a variable. You can use this string anywhere in the program using the name `my_variable`.

```
my_variable = "My first variable!"
```

```
print(my_variable)
```

In [27]:
```
my_variable = "My first variable!"
print(my_variable)
print(my_variable)
print(my_variable)
my_variable = 7
print(my_variable)
my_variable + my_variable
```

```
My first variable!
My first variable!
My first variable!
7
```

Out[27]:
```
14
```

You can also directly use literals anywhere in your program, as in the Hello world program. It is often more convenient to use variables for more sophiscated program.

Types

Each variable and literal has a **type**. It can be checked with the function `type()`.

```
type(my_variable)
type(300)
type(300.0)
```

In [28]:
```
type(300.0)
```

Out[28]:
```
float
```

In [29]:
```
type('Hello world')
```

Out[29]:
```
str
```

In [30]:
```
type(my_variable)
```

Out[30]:
```
int
```

Note: Only the result of the last line is displayed in Jupyter Notebook unless `print()` function is used. If the last line of the cell assigns something (`x = ...`), nothing is displayed.

In [31]:
```
my_variable1 = 100
my_variable2 = 10
print(my_variable1 + my_variable2)
type(my_variable1 + my_variable2)
```

```
110
```

Out[31]:
```
int
```

In [32]:
```
my_variable3 = 10.0  # float
print(my_variable1 + my_variable3) # what is the data type?
type(my_variable1 + my_variable3)
```

```
110.0
```

Out[32]:
```
float
```

Exercise.

Run the following code.

```
x = 5
x + 5
print(x)
```

In [33]:
```
x = 5
x + 5
print(x)
```

```
5
```

Why did not `x` change?

You can *reassign* a variable using the existing value as following:

```
x = x + 5
print(x)
```

In [34]:
```
x = x + 5
print(x)
```

```
10
```

Operator `+=` is for adding and reassign: `x += 5` is equivalent to `x = x + 5`. The same goes to `-=`, `*=`, `/=`, etc.

In [36]:
```
# x += 5 # x = x+5
# print(x)
x1 = x + 5
print(x1)
```

```
20
```

String operations and methods

Strings can be concatenated using the `+` sign:

```
"Hello" + ", world!"
```

In [37]:
```
"Hello" + ", world!"
```

Out[37]:
```
'Hello, world!'
```

or you can use a `.join()` **method**:

```
"".join(["Hello", ", world"])
```

Method is like function but dedicated to data of specific type. In this case, the `join` method of an empty string (`""`) is called to concatenate the strings in a **list** with the empty string as a delimiter.

List is a basic data structure in Python to represent a collection of items, defined by a pair of square brackets (`[]`). We will get back to it tomorrow.

In [38]:
```
print("".join(["Hello", ", world"]))
type("".join(["Hello", ", world"]))
```

```
Hello, world
```

Out[38]:
```
str
```

You can also format a string by replacing curly brackets (`{}`) with a value using a `.format()` method.

```
print("{} + {} = {}".format(2, 2, 2 + 2))
print("Discount: ${val:.2f}".format(val=2))
```

In [43]:
```
#print("{} + {} = {}".format(2, 2, 2+2))
print("Discount: ${val:.2f}".format(val=2.345))
```

```
Discount: $2.35
```

White spaces at the beginning or the end of the string can be removed using `.strip()` method.

```
"   asdfasdfascv   ".strip()
```

You can also count the number of occurrences of a substring using `.count()` method:

```
"   asdfasdfascv   ".count('as')
```

In [44]:
```
print("   asdfasdfascv   ")
print("   asdfasdfascv   ".strip())
```

```
   asdfasdfascv
asdfasdfascv
```

In [45]:
```
"   asdfasdfascv   ".count("as")
```

Out[45]:
```
3
```

Exercise (5 min).

1) Create and print variables:

```
s1 = "\t <x> <y> <z> \n"
s2 = "\n \n <x1> <y1> \n"
```

replace `<x>`, `<y>`, `<z>`, `<x1>`, `<y1>` is your own word.

Hint: `s = "a b"` - declaring variable s of string type.

In [46]:
```
s1 = "\t cats dogs pigs \n"
s2 = "\n \n CATS DOGS \n"
print(s1)
print(s2)
```

```
         cats dogs pigs



 CATS DOGS

```

2) "Strip" `s1` and `s2` and save them in variables `s3` and `s4`

Hint: `s.strip()` - remove white spaces from beginning and ending of `s`.

In [47]:
```
s3 = s1.strip()
s4 = s2.strip()
print(s3)
print(s4)
```

```
cats dogs pigs
CATS DOGS
```

3) Count the number of spaces in `s1` and `s3` using the `.count()` method.

Hint: `s.count('a')` - find the number of appearance of substring `'a'` in `s`.

In [48]:
```
print(s1.count(" "))
print(s3.count(" "))
```

```
4
2
```

4) Concatenate `s1` and `s2`, then print it.

In [49]:
```
print(s1 + s2)
```

```
         cats dogs pigs


 CATS DOGS

```

5) Concatenate `s3` and `s4`, then print it.

In [50]:
```
print(s3 +s4)
```

```
cats dogs pigsCATS DOGS
```

Quiz

What is the output of the code?

```
z = 5
z + 11
print(z)
```

- a. 5
- b. 11
- c. 16

In [51]:
```
z = 5
z + 11
print(z)
```
```
5
```

Quiz

What is the output of the code?

```
x = 41
z = x + 11
x = 80
print(z)
```

- a. 41
- b. 52
- c. 80
- d. 91

In [52]:
```
x = 41
z = x + 11
x = 80
print(z)
```
```
52
```

Exercise.

Compute the volume and surface area of a sphere with radius 6.

Hint. $V = \frac{4}{3}\pi r^3$ for the volume V, and $A = 4\pi r^2$ for area A, where r is the radius.

In [53]:
```
r = 6
pi = 3.14
V = (4/3)*pi*r**3
print(V)
A = 4*pi*r**2
print(A)
```
```
904.3199999999999
452.16
```

Casting

Sometimes, one may need to convert values between types. For example, it is impossible to concatenate a string and a number with a `+` operator. Numbers can be cast to string using `str()` function.

Likewise, one may convert a value using `int()` and `float()`.

```
"Discount: $" + 2 # This is an error.
```

In [54]:
```
"Discount: $" + 2
```
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[54], line 1
----> 1 "Discount: $" + 2

TypeError: can only concatenate str (not "int") to str
```

```
"Discount: $" + str(2)
```

In [56]:
```
"Discount: $" + str(2)
"Discount: $" + "2"
```
Out[56]:
```
'Discount: $2'
```

Exercise.

- Create one variable of string type and one variable of float type.
- Try to concatenate your variables.
- Convert float to string using `str()` and concatenate the variables again.

In [58]:
```
s = "String data"
f = 25.0
#print(s + f)
print(s + " " + str(f))
```
```
String data 25.0
```

Input and output

Let's try some interactive component.

`input()` is a function that reads data from user's input as a string.

```
input()
```

In [59]:
```
input()
```
```
abc
```
Out[59]:
```
'abc'
```

`input()` can redirect user's input to variables:

```
user_input = input()
```

In [60]:
```
user_input = input()
```
```
Hello!
```

In [61]:
```
print(user_input)
```
```
Hello!
```

`input()` can "ask questions":

```
user_mood = input("How are you today?")
```

In [63]:
```
user_mood = input("How are you today?")
print(user_mood)
```
```
How are you today?Good!
Good!
```

Reading numbers

Try running the following cells:

In [64]:
```
user_int = input("Input any integer: ")
print(2 * user_int)
```
```
Input any integer: 2
22
```

In [65]:
```
user_int = input("Input any integer: ")
print(user_int / 2)
```
```
Input any integer: 2
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[65], line 2
      1 user_int = input("Input any integer: ")
----> 2 print(user_int / 2)

TypeError: unsupported operand type(s) for /: 'str' and 'int'
```

`input()` reads in a string. What if we want to get a number from a user?

We can convert it to a number with `int()` or `float()` function.

You may try:

```
user_int = int(input("Input any integer: "))
print(2 * user_int)
```

In [66]:
```
user_int = int(input("Input any integer: "))
print(2 * user_int)
```

```
Input any integer: 2
4
```

Writing output

`print()` can output variables of different types to display, if they are given as separate parameters.

```
print(2)
print(2, ' + ', 2, ' = ', 4)
a = 2 * 2
print('two times two equals', a)
```

In [67]:
```
print(2)
```

```
2
```

In [68]:
```
print(2, ' + ', 2, " = ", 4)
```

```
2  +  2  =  4
```

In [69]:
```
a = 2 * 2
print("two times two equals", a)
```

```
two times two equals 4
```

As we did before, we can also use formatting:

```
print("{} + {} = {}".format(2, 2, 2 + 2))
```

In [70]:
```
print("{} + {} = {}".format(2, 2, 2 + 2))
```

```
2 + 2 = 4
```

As discussed before, floats and integers can be converted to string:

```
a = 2.5
print("'a' has type: ", type(a))
b = str(a)
print("'b' has type: ", type(b))
```

In [71]:
```
a = 2.5
print("'a' has type: ", type(a))
b = str(a)
print("'b' has type: ", type(b))
```

```
'a' has type:  <class 'float'>
'b' has type:  <class 'str'>
```

Exercise.

Write a program that:

1. Reads user's sentence
2. Counts the number of `WORDs` in the sentence and stores as a variable
3. Prints the number of `WORDs` to the user.

Hint: Use `input()` and `print()` functions. Count the number of spaces with the `.count()` method.

In [72]:
```
# write your code here
user_input = input("Input a sentence: ").strip()
space_count = user_input.count(" ")
print(space_count)
num_words = space_count + 1
print(num_words)
```

```
Input a sentence: How are  you?
3
4
```

Exercise.

Write a program that:

1. Reads a radius of a sphere from the user;
2. Computes volume and surface area of a sphere and stores them in variables;
3. Prints volume and surface area for the user.

Hint: $V = \frac{4}{3}\pi r^3$ for the volume V, and $A = 4\pi r^2$ for area A, where r is the radius.

In [74]:
```
pi = 3.14159265358979
r = float(input("Input radius: "))
V = (4 / 3) * pi * (r ** 3)
print(V)
print("{v:.4f}".format(v=V))
#print(v)
A = 4 * pi * r ** 2
print(A)
# write your code here
```

```
Input radius: 6.2
998.30599192633
998.3060
483.05128641596616
```

Exercise. Celcius to Fahrenheit converter

"Ugh, It's been so hot out lately, why does it have to be 32 degrees..."

"32 degrees? That's freezing!"

Input: A single floating-point number, `C`.

Output: Output a single floating-point number, `F`, the temperature in degrees Fahrenheit which is equivalent to `C` degrees Celsius.

$$F = \frac{9}{5}C + 32$$

In [75]:
```
C = float(input("Input degrees celcius: "))
F = (9 / 5) * C + 32
print(F)
```

```
Input degrees celcius: 32
89.6
```

Later in this workshop, we will learn how to formally write it as a function.

Booleans and logical expressions / Decisions

Booleans

There is a separate data type for Yes or No, called **boolean**.

The two possible values are: `True` or `False`.

```
print(type(True))
```

In [76]:
```
print(type(True))
```

```
<class 'bool'>
```

There are relational operations that return booleans:

- `x < y` -- is `x` less than `y`?
- `x <= y` -- is `x` less or equal to `y`?
- `x > y` -- is `x` greater than `y`?

- `x >= y` -- is `x` greater or equal to `y`?
- `x == y` -- is `x` equal to `y`? (not to be confused with the assignment operator `=`)
- `x != y` -- is `x` not equal to `y`?

```
10 < 15
12 > 12
"ab" >= "bb" # in alphabetical order, does "ab" come after or is same as "bb"?
```

In [77]:
```
10 < 15
```
Out[77]:
```
True
```

In [78]:
```
12 > 12
```
Out[78]:
```
False
```

In [84]:
```
"ab" > "Ab"
```
Out[84]:
```
True
```

Exercise.

1. Create and initialize two variables of numerical type `x` and `y`.
2. Create a variable `b` that is equal to `x <op> y`, where `<op>` is a relational operation such as `>`, `<`, `==`, `!=`, etc.
3. print variable `b`.
4. print type of the variable `b`.

In [87]:
```
x = 3.2
y = 4.0
b = x < y
print(b)
print(type(b))
```
```
True
<class 'bool'>
```

Quiz.

What is the output of the following code?

```
x = 12
b = (x == 5)
print(b)
```

- a) 5
- b) 12
- c) True
- d) False

In [88]:
```
x = 12
b = (x == 5)
print(b)
```
```
False
```

If statement

If statement can alternate the program flow using booleans.

```
if <logical_statement>:
    "this part will be executed if logical_statement is True"
elif <logical_statement_2>: # (optional)
    "this part will be executed if logical_statement is False and logical_statement_2 is True"
else: # (optional)
    "this part will be executed if logical_statement and logical_statement_2 are both False"
```

In Python, code blocks for control flow is defined by **indentation**, rather than a set of brackets or `end` keywords. Be careful with indentation levels. It is recommended to use four-space indentation, and some code editors will automatically do that for you.

```
eagles_total_score = 90
tigers_total_score = 97
if eagles_total_score > tigers_total_score:
    print("Eagles win!")
```

In [89]:
```
eagles_total_score = 90
tigers_total_score = 97
if eagles_total_score > tigers_total_score:
    print("Eagles win!")
```

```
eagles_total_score = 90
tigers_total_score = 97
if eagles_total_score > tigers_total_score:
    print("Eagles win!")
else:
    print("Tigers win!")
```

In [92]:
```
eagles_total_score = 90
tigers_total_score = 90
if eagles_total_score > tigers_total_score:
    print("Eagles win!")
# else:
#     print("Tigers win!")
elif eagles_total_score < tigers_total_score:
    print("Tigers win!")
else:
    print("Ties")
```

```
Ties
```

Exercise (3 min).

Write a program that:

1. Accepts the user's input for Eagles score
2. Accepts the user's input for Tigers score
3. Prints the winning team

In [93]:
```
eagles_total_score = int(input('Eagles: '))
tigers_total_score = int(input('Tigers: '))
if eagles_total_score > tigers_total_score:
    print("Eagles win!")
elif eagles_total_score < tigers_total_score:
    print("Tigers win!")
else:
    print('Ties')
```

```
Eagles: 90
Tigers: 95
Tigers win!
```

Unit tests.

- To maintain the quality of the code, each line of the program should work reliably and without errors.
- The concept of unit testing allows software developers to make sure that each part of the code is working correctly.
- A set of trivial tasks with known answers that allow to prove the correctness of the code is called a set of unit tests.

You need to check with several of your inputs if the code works correctly. Does it work with Eagles 40 points and Tigers 40 points?

In [94]:
```
eagles_total_score = int(input())
tigers_total_score = int(input())
if eagles_total_score > tigers_total_score:
    print("Eagles win!")
elif eagles_total_score < tigers_total_score:
    print("Tigers win!")
else:
    print("Ties")
```

```
90
90
Ties
```

Since indentations matter in Python, there is a keyword `pass` that does nothing in the code block.

In [95]:
```
a = 3
b = 3
if a > b:
    print("a is greater")
elif a == b:
    pass
else:
    print("b is greater")
```

Quiz.

What does this program print?

```
a = 17
if a > 12:
    a = 1
if a <= 12:
    a = 3
else:
    a = 2
print(a)
```

- a. 1
- b. 2
- c. 3
- d. 12
- e. 17

In [97]:
```
a = 17
if a > 12:
a = 1  # True
# if a <= 12:
#     a = 3 # True
# else:
#     a = 2
# print(a)
```

```
  Cell In[97], line 3
    a = 1  # True
    ^
IndentationError: expected an indented block after 'if' statement on line 2
```

Boolean Operators: or, and, not

a	b	a or b	a and b
False	False	False	False
False	True	True	False
True	False	True	False
True	True	True	True

a	not a
False	True
True	False

- `and` has higher priority than `or`
- `not` has higher priority than `and`
- `()` can be used to change the priority

In [98]:
```
True and False
```

Out[98]:
```
False
```

In [99]:
```
True or False
```

Out[99]:
```
True
```

In [ ]:
```
True or True and False
```

In [ ]:
```
True or (True and False)
```

In [ ]:
```
(True or True) and False
```

In [ ]:
```
not True
```

In [ ]:
```
(5 > 4) and (5 <= 4)
```

In [ ]:
```
(5 > 4) or (5 <= 4)
```

In [ ]:
```
(5 > 4) or (5 <= 4) and (5 <= 4)
```

In [ ]:
```
((5 > 4) or (5 <= 4)) and (5 <= 4)
```

In [ ]:
```
not True and False
```

In [ ]:
```
(not True) and False
```

In [ ]:
```
not (True and False)
```

Exercise.

Write a program that checks if a number from a user's input is divisible by 3, 5 or 7, but NOT by 15, 21, 35 or 105.

In [100…
```
input_number = int(input('Input a number: '))
if (input_number % 3 == 0) or (input_number % 5 == 0) or (input_number % 7 == 0):
    if (input_number % 15 != 0) or (input_number % 21 != 0) or (input_number % 35 != 0) or (input_number % 1
        print('divisible')
    else:
        pass
```

```
Input a number: 105
```

Exercise.

Write a program that checks if the year from a user's input is a leap year.

Hint: To check if a year is a leap year, divide the year by 4. If it is fully divisible by 4, it is a leap year. For example, the year 2016 is divisible 4, so it is a leap year, whereas, 2015 is not. However, century years like 300, 700, 1900, 2000 need to be divided by 400 to check whether they are leap years or not. While century years not divisible by 400 is not a leap year, century years divisible by 400 is a leap year.

In [101…
```
year = int(input("Year:" ))
if year % 4 == 0:
    if year % 100 == 0:
        if year % 400 == 0:
            print("Leap year")
        else:
            print("Not leap year")
    else:
        print("Leap year")
else:
    print("Not leap year")
```

```
Year:2012
Leap year
```

Assignment 1

The progressive income tax rate is mandated as follows:

Taxable Income	Rate (%)
First $5,000	0
Next $10,000	7

Taxable Income	Rate (%)
Next $15,000	15
The remaining	25

For example, suppose that the taxable income is $70,000:

The income tax payable is $\$5,000 \times 0\% + \$10,000 \times 7\% + \$15,000 \times 15\% + \$40,000 \times 25\%$. Write a program that reads the taxable income from user. The program shall calculate the income tax payable.

```
In [ ]:
```

Assignment 2

Write a program to examine if the user is providing correct password "UCLAbioInfo". If the wrong attempts go over 3 times, print "The account is locked".

```
In [ ]:
```

Assignment 3

A vending machine is selling 2 products, coke and chips, with the price,

Product	Price
Coke	1.99
Chip	2.35
Combo	10% off

*Combo means the order include both

Write a program that

- Asks how many cokes the customer is purchasing
- Asks how many chips the customer is purchasing
- Display the final price

```
In [ ]:
```

Additional Tasks

1. https://dmoj.ca/problem/ccc06j1
2. https://dmoj.ca/problem/ccc15j1
3. https://dmoj.ca/problem/dmopc16c1p0