計算物理概論

Introduction to Computational Physics (PHYS290000)

Lecture 3: Basic Python (part 1)

Instructor: 潘國全

kuochuan.pan@gapp.nthu.edu.tw

Last week

1. Installed python via Anaconda
2. Installed VS Code
3. Created a virtual python environment for this course
4. Installed Jupyter notebook
5. Hello world program in python (.py and .ipynb)
6. (Not talked yet) Version control with Git

Today’s plan

1. Basic Python Programming:
2. Python syntax
3. Variables
4. Collections
5. Control Flow
6. Functions

Basic Python Programming

Using Python as a Calculator

One common way to use python is to use it as a calculator.
You could either create a juoyter notebook (like this file) or type `python` to enter the interactive mode.

```
[2]  2 + 2 * 5 / 4 - 1 + (5 - 3)*2
     ✓ 0.0s                                                    Python
```

```
...  7.5
```

Using Python as a Calculator

The integer numbers (e.g. `1`, `2`, `100`) have type `int`. The ones with a fractional part (e.g. `2.0`, `3.14`, `1.989e33`) have type `float`. We will see more about numeric types later.

Division (`/`) always returns a float. To do floor division and get an integer result, you could use the `//` operator; to calculate the remainder you could use `%`:

```
print(21/8)   # classic division returns a float
print(21//8)  # floor division returns an int and discards thefractional part
print(21%8)   # the % operator returns the remainder (int)
print(8*2+5)  # floored quotient * divisor + remainder
```

[6] ✓ 0.0s Python

```
2.625
2
5
21
```

Using Python as a Calculator

The equal sign (=) is used to assign a valure to a variable.

```
c  = 2.99792458e10 # speed of the light [cgs]
k  = 1.38e-16      # Boltzmann constant [cgs]
h  = 6.626e-27     # Planck constant    [cgs]
pi = 3.1415926535  # pi

sigma = (2*pi**5/15)*(k**4)/(c**2 * h**3) # Stefan-Boltzmann constant
print("The Stefan-Boltzmann constant is:", sigma, " [erg/s/cm^2/K^4]")
```

[12] ✓ 0.0s Python

```
The Stefan-Boltzmann constant is: 5.659899831296445e-05  [erg/s/cm^2/K^4]
```

Using Python as a Calculator

You could also use the `math` library to do more advanced calculations.

```
import math
angle  = math.pi/4
answer = math.sqrt(math.sin(angle)**2 + math.cos(angle)**2)
print(answer)
```

[8] Python

```
1.0
```

Note

Only use the `math` library when doing simple scalr calculations.
For scientific computations (especially when handling vector, tensor, or more complicated computations), use `numpy` and do not use `math`.

We will learn `numpy` in this semester.

More on the math library

1. RTFM: https://docs.python.org/3/library/math.html

`acos(x)`	returns the arccosine of `x` ($\cos^{-1} x$)
`asin(x)`	returns the arcsine of `x` ($\sin^{-1} x$)
`atan(x)`	returns the arctangent of `x` ($\tan^{-1} x$)
`atan2(y, x)`	returns the arctangent of `y/x` ($\tan^{-1}(y/x)$)
`cos(x)`	returns the cosine of `x` radians ($\cos x$)
`degrees(x)`	returns the number of degrees in `x` radians
`exp(x)`	returns $e^{\mathtt{x}}$
`log(x, [b])`	returns the logarithm base `b` of `x` ($\log_{\mathtt{b}} \mathtt{x}$); if `b` is omitted, returns the natural logarithm of `x` ($\ln \mathtt{x}$)
`radians(x)`	returns the number of radians in `x` degrees
`sin(x)`	returns the sine of `x` radians ($\sin x$)
`sqrt(x)`	returns the square root of `x` ($\sqrt{x}$)
`tan(x)`	returns the tangent of `x` radians ($\tan x$)
`e`	the value of e (Euler's number), the base of the natural logarithm
`pi`	the value of π

Maximum, minumum, and absolute value are build-in functions:

max(a,b), min(a,b), abs(a)

Basic Python Syntax (Identifiers)

1. A python identifier is a name use to identify a variable, function, class, module or other objects. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).

2. Python does not allow punctuation characters such as @, $, and % within identifiers.

3. Python is a case sensitive programming language. “Physics” and “physics” are two different identifiers in Python.

Basic Python Syntax (Identifiers)

```python
student_id = 123
age        = 19
height     = 175.8
name       = "你的名字"
_isStudent = True
print(name)  # print the variable
del name     # delete a variable
print(name)
```

[23] ⊗ 0.2s Python

```
你的名字
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[23], line 8
      6 print(name)  # print the variable
      7 del name     # delete a variable
----> 8 print(name)

NameError: name 'name' is not defined
```

Basic Python Syntax (Naming conventions)

1. Naming conventions: Python identifiers start with a lowercase letter, except Class names. [See PEP 8]

2. Starting an identifier with a single leading underscore indicates that the identifier is a private identifier (to be used only in the class, but not forced). [See PEP 8]

3. Use all uppercase letters to define a constant.

4. Starting an identifier with two leading underscores indicates a strong private identifier (will returns an error). [See PEP 8]

5. If the strong private identifier also ends with two trailing underscores, the identifier is a language-defined special name.

Basic Python Syntax (reserved words)

1. These words are reserved. You cannot use them as identifier names.

and	as	assert
break	class	continue
def	del	elif
else	except	False
finally	for	from
global	if	import
in	is	lambda
None	nonlocal	not
or	pass	raise
return	True	try
while	with	yield

Basic Python Syntax (Multi-line statements)

Statements in Python typically end with a new line. Python does, however, allow the use of the line continuation character (\) to denote that the line should continue.

```
variable = 1 + 2 + 3 + 4 + 5\
         + 6 + 7 + 8 + 9 + 10\
         + 11 + 12 + 13
```

[6] ✓ 0.0s Python

Statements contained within the [], {}, or () brackets do not need to use the line continuation character. For example following statement works well in Python.

```
days = ['Monday', 'Tuesday', 'Wednesday',…
```

[5] ✓ 0.0s Python

Python Indentation [See PEP 8].

1. Use 4 spaces per indentation level [See PEP 8].
2. Extra level of indentation is ok for clarity.
3. Limit all lines to a maximum of 79 characters.
4. Line break before a binary operator.

```
[11]  ✓  0.0s                                                    Python

forces = (gravitational_force
          + electromagnestim_force
          + weak_force
          + strong_force )
```

Exercise 1:

RTFM: https://docs.python.org/3/library/math.html

The electron fraction at nuclear density can be fitted by a parametrized equation (eq 1). Table 1 shows the fitted parameters for two high-density models, N13 and G15. Calculate the electron fraction at rho=1.e13 g/cm^3 for these two models.

TABLE 1

PARAMETERS FOR THE FITTING FORMULA

	N13		G15	
PARAMETER	ρ_i (g cm^{-3})	Y_i	ρ_i (g cm^{-3})	Y_i
$i = 1$	2×10^7	0.5	3×10^7	0.5
$i = 2$	2×10^{13}	0.285	2×10^{13}	0.278
Y_c	...	0.035	...	0.035

The fitting formula reads

$$x(\rho) = \max\left[-1, \min\left(1, \frac{2\log\rho - \log\rho_2 - \log\rho_1}{\log\rho_2 - \log\rho_1}\right)\right],$$

$$Y_e(x) = \frac{1}{2}(Y_2 + Y_1) + \frac{x}{2}(Y_2 - Y_1) + Y_c[1 - |x| + 4|x|(|x| - 1/2)(|x| - 1)]. \quad (1)$$

Quotations

Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals, as long as the same type of quote starts and ends the string.

The triple quotes are used to span the string across multiple lines.

```
word = 'physics'

sentence = "This is a sentence."

paragraph = """ This is a paragrah. It is made up
    of multiple lines and sentences.
    You could use '\\n'\n to break lines as well. """
print(paragraph)
```

[16] ✓ 0.0s Python

```
This is a paragrah. It is made up
   of multiple lines and sentences.
    You could use '\n'
to break lines as well.
```

Comments in Python

Comments

```
# First comment
force = mass * acceleration # second comment
'''
This is a triple-quoted string but
in here, can be used as a comment as well.
'''
```

[20] ✓ 0.0s Python

```
'\nThis is a triple-quoted string but  \nin here, can be used as a comment as well.\n'
```

Python Variables

```
n     = 100     # integer variable
a     = 123.45  # float variable
c     = 3+5j    # complex variable
s     = "A string"  # a string
isFun = True    # A boolean type
```

[ ]

Python

String Data Type

```
hello = "Hello World! Python rocks!"
print(hello)      # print the entire strong
print(hello[0])   # print the first character
print(hello[2:5]) # prints characters starting from 3rd to 5th
print(hello * 2)  # prints the string two times
print(hello + " Physics is func!") # prints concatenated string
print(hello.split()) # split the string by spaces
```

[25] ✓ 0.0s Python

```
Hello World! Python rocks!
H
llo
Hello World! Python rocks!Hello World! Python rocks!
Hello World! Python rocks! Physics is func!
['Hello', 'World!', 'Python', 'rocks!']
```

Python Collections

Lists	Tuples	Sets	Dictionaries
A list is a collection of *ordered* data.	A tuple is an *ordered* collection of data.	A set is an *unordered* collection.	A dictionary is an *unordered* collection of data that stores data in key-value pairs.

```
list1 = [1,2.0,"apple","5",6]
list2 = [] # empty list
list3 = list((1,2,3,4,5))
list4 = [1,2,3,[1,2,3],(4,5,6)]
```

```
tuple1 = (1,2,3,"comp","phys")
tuple2 = () # empty tuple
tuple3 = tuple((1,2,3,4,5))
tuple4 = ((1,2,3),4,5,[6,7])
```

```
set1 = {1,2,3,"comp","phys"}
set2 = set((1,2,3,4,5))
set3 = set() # empty set
set4 = {(1,2,3),(4,5,6)}
```

```
dict1 = {"key1":"value1","key2":2,"key3":[1,2,3]}
dict2 = {} # empty dictionary
dict3 = dict({"key1":"value1",2:"value2"})
```

Python Collections

Lists are *mutable*.	Tuples are *immutable*.	Sets are *mutable* and have *no duplicate elements*.	Dictionaries are mutable and keys do not allow duplicates.

```
list1 = [1,2.0,"apple","5",6]
print(len(list1)) # length
print(list1[0])   # 1st element
list1[0] = 999
print(list1)
print(list1[-1])  # last element
print(list1[2:4]) # 3rd to 4th ele
print(list1+list3) # concatenation
```

✓ 0.0s

```
5
1
[999, 2.0, 'apple', '5', 6]
6
['apple', '5']
[999, 2.0, 'apple', '5', 6, 1, 2, 3, 4, 5]
```

Python Collections

Lists are *mutable*.	Tuples are *immutable*.	Sets are *mutable* and have *no duplicate elements*.	Dictionaries are mutable and keys do not allow duplicates.

```
tuple1 = (1,2,3,"comp","phys")
print(len(tuple1)) # length
print(tuple1[0])   # 1st element
print(tuple1[1:3])
#tuple1[0] = 999   # won't work
```

Python Collections

Lists are *mutable*.	Tuples are *immutable*.	Sets are *mutable* and have *no duplicate elements*.	Dictionaries are mutable and keys do not allow duplicates.

```
set1 = {3,2,1,"comp","phys"}
print(set1)
set2 = {1,2,3,3,2,2,1}
print(set2)
# print(set2[0]) # set is unordered
```

Python Collections

Lists are *mutable*.	Tuples are *immutable*.	Sets are *mutable* and have *no duplicate elements*.	Dictionaries are mutable and keys do not allow duplicates.

```
dict1 = {"key1":"value1","key2":2,"key3":[1,2,3]}
print(dict1.keys())
print(dict1.values())
dict1["key1"] = "updated value"
dict1["key4"] = "new value"
print(dict1)
```

✓ 0.0s

```
dict_keys(['key1', 'key2', 'key3'])
dict_values(['value1', 2, [1, 2, 3]])
{'key1': 'updated value', 'key2': 2, 'key3': [1, 2, 3], 'key4': 'new value'}
```

Python Collections

Lists are *mutable*.	Tuples are *immutable*.	Sets are *mutable* and have *no duplicate elements*.	Dictionaries are mutable and keys do not allow duplicates.

```
list1.append("new")    # append enw value
#tuple1.append("new") # tuple is immutable
set1.add("new")        # add new element
dict1.update({"new_key":"value"}) # add a new key

list1.pop()          # remove the last value
# tuple1.pop()       # tuple is immutable
set1.pop()           # remove one random element
dict1.pop("key1")    # remove the item with key="key1"
```

Python Collections: Index

```
list1 = [1,5,2,5,7,"physics"]
print(list1.index(5))   # returns the index value of the element at its first occurence
print(list1.index(5,3)) # returns the index value of the element from the particular sta

tuple1 = (1,5,2,5,7,"physics")
print(tuple1.index(5))
print(tuple1.index(5,2))

# set1.index()    # set is unordered
# dict1.index()   # dictionary is unordered
print(dict1.get("key1"))
print(dict1["key1"])
```

✓ 0.0s Python

```
1
3
1
3
updated value
updated value
```

Python Collections: More

1. There more methods for the list, tuple, set, and dict data type

2. See here: https://docs.python.org/3/tutorial/datastructures.html

Control Flow: “if” statement

```
x = 10
if x < 0:
    print("x is negative.")
elif x==0:
    print("x is zero.")
elif x==1:
    print("x is one.")
else:
    print("x is ", x)
```

[1] ✓ 0.0s

```
x is  10
```

```
if (a > 3) and (b < 5):
    print(" a > 3 and b < 5")
elif (a==2 or a==3 or a==6):
    print("logical or")
elif (b != 4):
    print("b is not equal to 4")
```

Python Comparison Operators

Python comparison operators compare the values on either sides of them and decide the relation among them. They are also called relational operators. These operators are equal, not equal, greater than, less than, greater than or equal to and less than or equal to.

Operator	Name	Example
==	Equal	4 == 5 is not true.
!=	Not Equal	4 != 5 is true.
>	Greater Than	4 > 5 is not true.
<	Less Than	4 < 5 is true.
>=	Greater than or Equal to	4 >= 5 is not true.
<=	Less than or Equal to	4 <= 5 is true.

Example

Control Flow: “for” statement

```
list1 = ["cat","dog","Einstein","cow"]
for item in list1:
    print(item)

tuple1 = (1,2,3,4,5)
for item in tuple1:
    print(item)

set1 = {"a",1,3,"b"}
for item in set1:
    print(item)

dict1 = {"a":1,"b":2,"c":3}
for key, value in dict1.items():
    print(key,value)

for i in range(5):
    print(i)

for i in range(0,10,2):
    print(i)
```

Control Flow: “for” statement

```
for n in range(2,10): # the first loop
    for x in range(2,n): # the second loop
        if n % x == 0:
            print(n, 'equals', x, ' x ' ,n//x)
            break # break the second loop
    else:
        # loop failed to finding a factor
        print(n, 'is a prime number.')
```

[15] ✓ 0.0s

```
2 is a prime number.
3 is a prime number.
4 equals 2  x  2
5 is a prime number.
6 equals 2  x  3
7 is a prime number.
8 equals 2  x  4
9 equals 3  x  3
```

```
for num in range(2,10):
    if num % 2:
        print('found an even number', num)
        continue # jump to the next index in the loop
    print('found an odd number', num)
```

[16] ✓ 0.0s

```
found an odd number 2
found an even number 3
found an odd number 4
found an even number 5
found an odd number 6
found an even number 7
found an odd number 8
found an even number 9
```

```
for n in range(10):
    pass # pass does nothing
```

Control Flow: “while” statement

```
counter = 0
while counter <= 10:
    print(counter)
    if counter ==12:
        break
    counter += 1 # counter = counter + 1
else:
    print("counter > 10.", counter)
```

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3

Control Flow: “match” statement


```
match status:
    case 400:
        print("Bad request")
    case 404:
        print("Page not found")
    case 418:
        print("I'm a teapot")
    case _:
        print("Something is wrong with the internet.")
```

Control Flow: “enumerate” statement

```
list1 = ["a","b","c","d"]
for i,value in enumerate(list1):
    print(i, value)
```

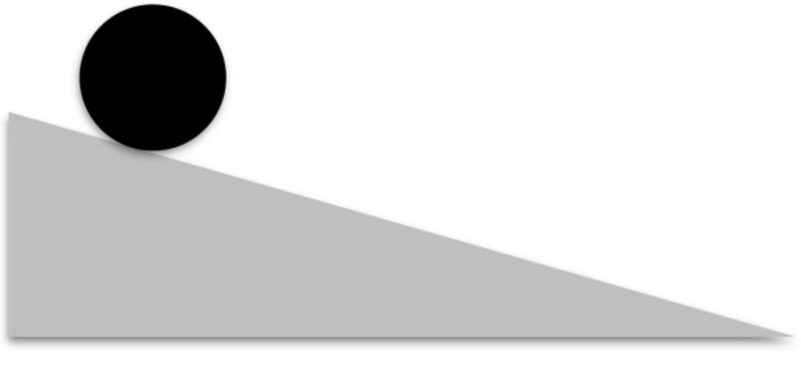
✓ 0.0s

```
0 a
1 b
2 c
3 d
```

```
for i in range(len(list1)):
    print(i, list1[i])
```

Exercise 2:

1. A ball (m = 1.5 kg) is released freely on a slope at height H, the height and velocity of the ball are measured in the right table.
2. Create three lists to store the data of the ball's time, height, and velocity.
3. Loop all the data and compute the total energy of the ball (kinetic + potential)
4. Store the total energy in another list
5. Use a loop to print out the value of total energy and compute the average.



Time [s],	Velocity [m/s],	Height [m]
0.004	0.013	2.004
0.215	0.679	1.987
0.417	1.306	1.988
0.605	1.920	1.862
0.813	2.638	1.684
1.003	3.236	1.573
1.209	3.897	1.327
1.411	4.516	1.070
1.602	5.010	0.768
1.805	5.803	0.416

Function

```
# This is a simple function
def hello():
    print("Hello world!")

# function could take argumen
def say(words):
    print(words)
    return # optional but better have it

# function with multiple arguments
def add(a,b):
    print(a,"+",b," = ", (a+b))
    return (a+b) # return the sum result

# invoke functions
hello()
say("Hello world!")
ans = add(1,2)
```

Add Markdown Cell

[1] ✓ 0.0s Python

```
Hello world!
Hello world!
1 + 2  =  3
```

Function: pass by reference vs value

All parameters (arguments) in the Python language are passed by reference. It means if you change what a parameter refers to within a function, the change also reflects back in the calling function.

```
mylist = [1,2,3,4]

def change(list):
    list.append(5) # mylist has passed into this function
    print("Values inside the function:", list)

change(mylist)
print("Values outside the function:", mylist)
```

✓ 0.0s

```
Values inside the function: [1, 2, 3, 4, 5]
Values outside the function: [1, 2, 3, 4, 5]
```

```
mylist = [1,2,3,4]

def change(list):
    list = [1,2,3,4,5] # This would assig new reference in list
    print("Values inside the function:", list)

change(mylist)
print("Values outside the function:", mylist)
```

✓ 0.1s

```
Values inside the function: [1, 2, 3, 4, 5]
Values outside the function: [1, 2, 3, 4]
```

Function: default arguments

```
def printinfo(name, age=18):
    print("Name: ", name)
    print("Age : ", age)

printinfo(name="Einstein", age=35)
printinfo(age=28, name="Maxwell")
printinfo(name="Newton")
```

[5] ✓ 0.0s

```
Name:  Einstein
Age :  35
Name:  Maxwell
Age :  28
Name:  Newton
Age :  18
```

Function: Arbitrary (keyword) arguments

```
def npc(name, *args, **kwargs):
    print("The NPC's name is ", name)
    for arg in args:
        print(arg)
    for kw in kwargs:
        print(kw, ":", kwargs[kw])

npc("Albert Einstein",
    "The most famous physicist.",
    age=35,
    hp=500,
    mp=1250,
    skills=("special relativity","relativity","photoelectric effect"),
    )
```

[6] ✓ 0.0s Python

```
The NPC's name is  Albert Einstein
The most famous physicist.
age : 35
hp : 500
mp : 1250
skills : ('special relativity', 'relativity', 'photoelectric effect')
```

Better Function with type hints (after v. 3.5)


```python
Vector = list[float]
def calculate_angular_momentum(mass: float,
                               position: Vector,
                               velocity: Vector) -> Vector:
    """
    This function calculate the angular momentum vector
    based on given mass, position, and velocity.

    L = m r x v

    Note: using `numpy` will be much easier.

    :param mass: the mass of this object
    :param position: the position vector
    :param velcity: the velocity vector
    :return: the angular momentum vector
    """
    ang_mom = [mass*(position[1]*velocity[2]-position[2]*velocity[1]),
               mass*(position[2]*velocity[0]-position[0]*velocity[2]),
               mass*(position[0]*velocity[1]-position[1]*velocity[0])]
    return ang_mom

mass     = 2.0
position = [1,2,3]
velocity = [4,5,6]
calculate_angular_momentum(mass=mass,position=position,velocity=velocity)
```

Note: The Python runtime does not enforce function and variable type annotations. They can be used by third party tools such as type checkers, IDEs, linters, etc.

Exercise 3:

1. Now, back to the first exercise.

2. Write a function that takes density, rho1, rho2, Y1, Y2, and Yc as an input arguments.

3. Compute the electron fraction Ye at density = 1e12, 1e13, and 1e14 for models N13 and G15.

TABLE 1

PARAMETERS FOR THE FITTING FORMULA

	N13		G15	
PARAMETER	ρ_i (g cm^{-3})	Y_i	ρ_i (g cm^{-3})	Y_i
$i = 1$	2×10^7	0.5	3×10^7	0.5
$i = 2$	2×10^{13}	0.285	2×10^{13}	0.278
Y_c	...	0.035	...	0.035

The fitting formula reads

$$x(\rho) = \max\left[-1, \min\left(1, \frac{2\log\rho - \log\rho_2 - \log\rho_1}{\log\rho_2 - \log\rho_1}\right)\right],$$

$$Y_e(x) = \frac{1}{2}(Y_2 + Y_1) + \frac{x}{2}(Y_2 - Y_1) + Y_c[1 - |x| + 4|x|(|x| - 1/2)(|x| - 1)]. \quad (1)$$

A python file (.py)

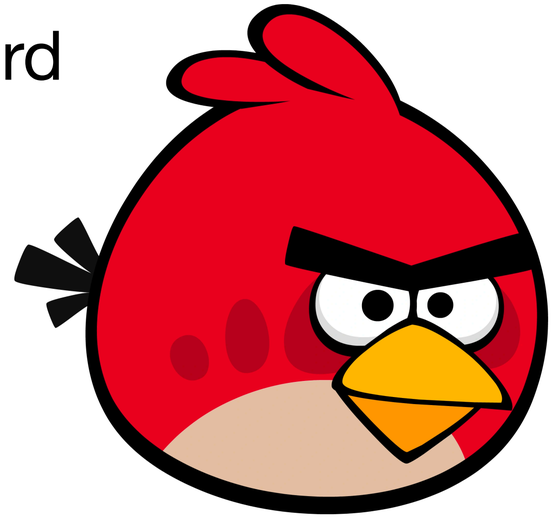
tut_05_python.py > ...

```python
"""

Here is the description of this file.
We should describe the usage of this file

Author: Your Name
Email: xxx@gapp.nthu.edu.tw
Copyright: Your Name
Liscence: ...<could be important>
Date: 2023.03.06

"""
def main():
    """
    This is the main function of this file


    """
    print("Enters the main function.")
    function1()

    return


def function1():
    """
    The main function might invoke this function.
    """
    print("function 1 invoked.")
    return


if __name__=='__main__':

    # if this file is run as the main file, it will
    # execute from here:

    main() # invoke the main function.
```

Exercise 4:

1. Assuming you are playing the angry bird game, you could set the initial angle and velocity of the angry bird.

2. If there is no air resistant, the trajectory of the angry bird has analytical solution



$$x = x_0 + v_0 \cos(\theta) t$$

$$y = y_0 + v_0 \sin(\theta) t + \frac{1}{2} g t^2$$

3. Set the initial position (x0, y0) to (0,0) and g = -9.81 m/s^2

Exercise 4:

1. Write a function to compute how long can the angry bird fly. The angry bird will stop when reaching the ground (y=0).

2. Write a function to compute the distantance that the angry bird can travel.

3. Assuming the pig located at x = 100 meter, write a python program that takes two inputs (angle and velocity) of the angry bird and check if you could hit the pig. If you hit the pig, you win and game finished; if not, continue to take new inputs. [hit: defined by the distance difference is less than 1m]

Exercise 4:

4. Input values can be taken by `val = input("Enter your values (seperated by a space):")`

5. Make sure your program will raise errors when the number of input values are not two (exception: if the input is “quit”).

6. The player could quit the game at any time when typing “quit”.

7. If the angry bird does not hit the pig, print a message to suggest adjustments (e.x. “the angry bird flew too far! Try lower the angle or decrease the speed.”)

Miniexam 3

1. Check google classroom
2. Write your solutions in .py files (do not use Jupyter notebook)
3. Please write comments and explanation of your answers
4. Your code should always have if __name__==‘__main__’:
5. Your code must able to executed by python your_code.py
6. Answer will be show up with the “print()” function.

Miniexam 3: programming

1. Back to exercise 4.

2. You to make an AI player to play this angry bird game (not true AI).

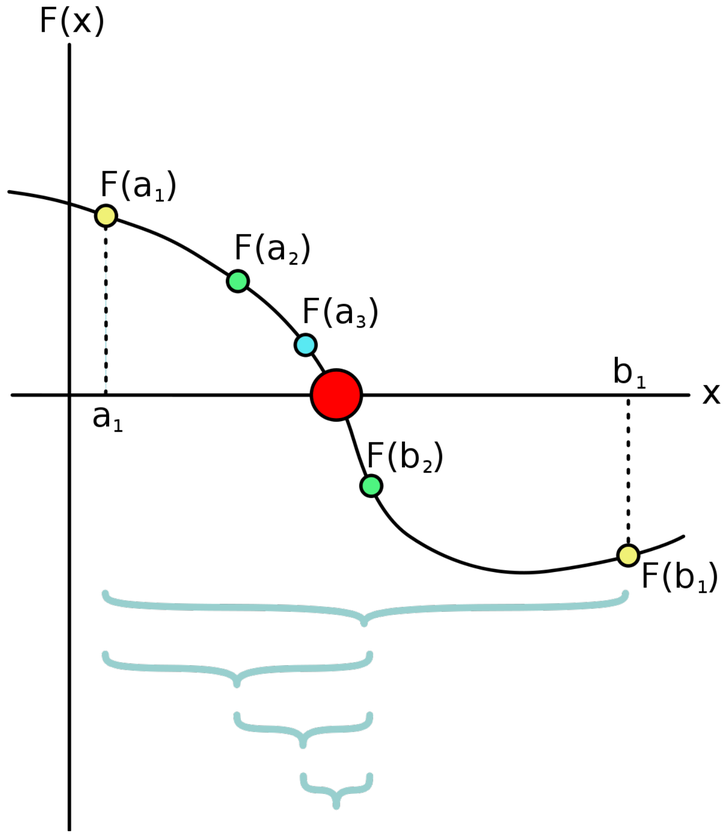
3. You still need to assign an initial angle (fix to 45 degree) and velocity of the angry bird, but the AI player will automatically adjust the "velocity" magnitude (fix the angle=45 degree).

4. The simplest way to adjust the "velocity" is to use the so-called "bisection" search.

5. Implement the bisection search in your angry bird game.

Miniexam 3: Bisection search

1. The algorithm of the bisection search can be described as searching the solution of f(x) = 0 (see below figure)



- “a” and “b” are two bracket that include the solution “x”.
- If f(a) and f(x) have different sign, update b.
- If f(a) and f(x) have the same sign, update a.
- x =(a+b)/2

Miniexam 3: programming

6. Make sure that your angry bird game will print out the intermediate and final solutions during the bisection search.

7. Re-do the problem, but set the “hit” criterion to dx <= 0.1 meter. Could your AI player hit the pig? How many trial did it spent?