Review Slides

COMP1021 final

Yongqiang Tian

yqtian@ust.hk

Note

- These slides covers important concepts in COMP1021
  - Common functions, Loop, Sequences, …

- These concepts are essential to midterm/exam
  - But only knowing these concepts is not sufficient
  - Practice: learn how to analyze problems

- These concepts are not complete
  - Feel free to add more as needed

If you find anything should be improved, please let me know!

I will acknowledge your contributions!

Basic concept

- Python – interpreted: each line of code is executed one by one
- In python, the index always starts from 0!
  - Not 1!
- Indentation (spaces before each line) is extremely important!

Input and Output

- `var = input("give me an input")`
  - note: `var` is always a string, not int/float
  - use `int(var)` as needed
- `print("Today it is windy!")`
- `print("Today it is windy!", var)`
  - Print strings and variables
- `print("Today it is windy!", end="??")`
  - `end` is used to control the ending after string.

Common functions

- `import random`
- `random.randint(1,10)`
  - `1 and 10 are inclusively`
- `type(a) will return the type of variables`

Turtle – basic concepts

- `import turtle`
- Three key elements:
  - Position, default is origin point (0,0)
  - Orientation: default is right
  - Pen: can be lifted or put down: default is down; has color and thickness
- `turtle.done()`
- `turtle.setup(width, height)`

Turtle – pen

- Pen has color and thickness
  - `turtle.width(width), turtle.color("red")`
- Pen can be lifted or put down; **nothing** is drawing after `up()`
  - `turtle.up() and turtle.down()`
  - Does not affect `turtle.dot()`
- `Color:`
  - Pen color: `turtle.color("red")`
  - Fill color: `turtle.fillcolor("red")`
  - Both: `turtle.color("red", "green")=>` Pen: `red;` fill: `green`
- `turtle.speed(speed):`
  - `1 is slow, 10 is fast, 0 is fastest`

Turtle – movement

- `turtle.forward(distance)`
- `turtle.backward(distance)`
  - Orientation is not changed when moving forwards/backwards!
  - Example of `distance: 100`
- `turtle.right(degree)`
- `turtle.left(degree)`
  - Change orientation of turtle
  - Example of `degree: 45/90/180/360`
- `turtle.goto(x, y)`
  - `x` and `y` are locations
  - `x` is horizontal, and `y` is vertical.

Turtle – drawing shapes

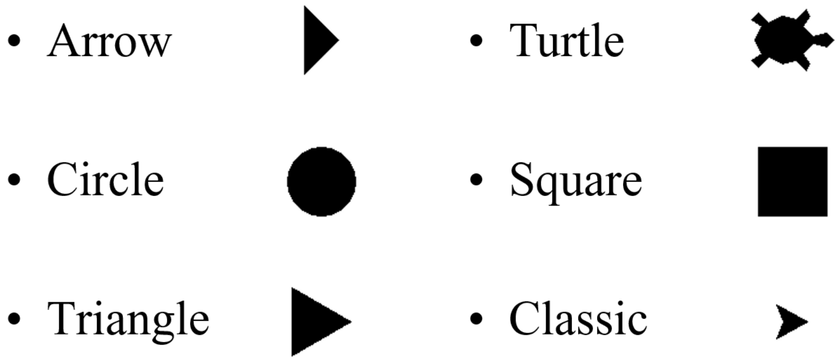
```
turtle.begin_fill()
xxxxx (code for drawing)
turtle.end_fill()
```

- `turtle.circle(radius)`
  - Center is radius pixel left of the turtle
  - counterclockwise if `radius>0`
- `turtle.circle(radius, degree)`
  - `Degree = 180` means half circle
- `turtle.clear():` clear the screen

Turtle - shape

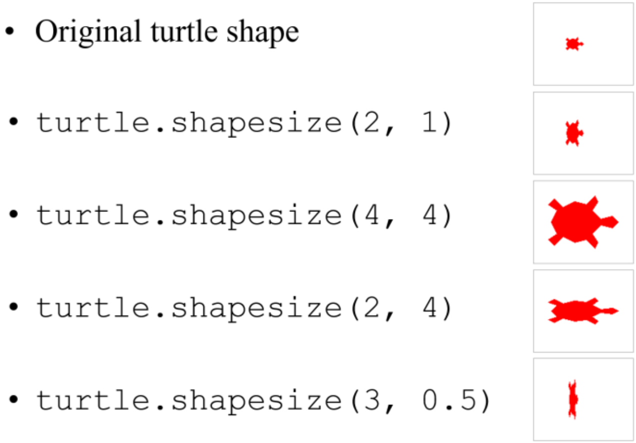
- `turtle.shape("classic")`

  ```
  turtle.addshape("ninja.gif")
  turtle.shape("ninja.gif")
  ```



- Arrow
- Turtle
- Circle
- Square
- Triangle
- Classic

- `turtle.shapesize(width_ratio, length ratio)`
  - `width_ratio = X` means the new width is X * original width



- Original turtle shape
- `turtle.shapesize(2, 1)`
- `turtle.shapesize(4, 4)`
- `turtle.shapesize(2, 4)`
- `turtle.shapesize(3, 0.5)`

Turtle – event handling

```
import turtle

def drawcircle(x, y):
    print(x,y)
    turtle.up()
    turtle.goto(0, -180)
    turtle.down()
    turtle.circle(250)

turtle.onclick( drawcircle )
turtle.done() # must!
```

When turtle is clicked, this function is called. The x and y where the turtle was clicked is passed to this function

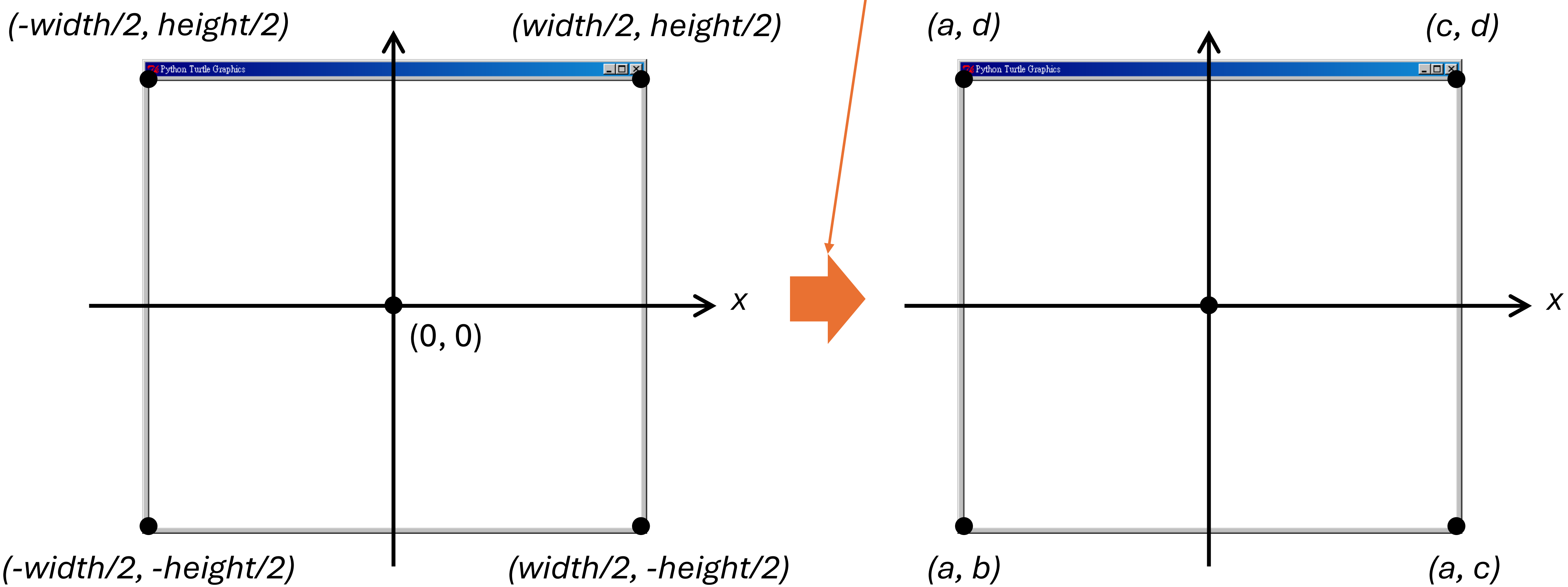
The `drawcircle` *function will be executed when the turtle is clicked on*

Turtle – event handling

- **`Event when we click/drag the turtle`**
- `turtle.onclick( drawcircle )`
- `turtle.ondrag( turtle.goto )`
- **`Event when we click screen other than turtle`**
- `turtle.onscreenclick( myfunction )`
- **`Event when we click keyboard`**
- `turtle.onkeypress( myfunction , "a")`
  - Remember `turtle.listen()`
  - `"a" can be "Up" "Down" "Left" "Right"`

Coordinate systems

```
turtle.setworldcoordinates(a,b,c,d)
a:min x, b:min y, c:max x, d:max y
```



Turtle Objects

- `newTurtle = turtle.Turtle()` will create a new turtle
  - `newTurtle` has the same function as the previous one
  - But different properties.
- `result = thisTurtle.xcor()` Get the x position value
- `result = thisTurtle.ycor()` Get the y position value
- `result = thisTurtle.position()` Get both x and y
- `result = thisTurtle.heading()` Get the turtle angle
- `result = thisTurtle.fillcolor()` Get the fill color
- `result = thisTurtle.speed()` Get the speed
- `result = thisTurtle.shape()` Get the shape

Turtle – other

- `turtle.hideturtle()`
- `turtle.write(string)`
- `turtle.write(string, font=("Arial", 20, "bold"))`
  - Write with specific font
- `turtle.dot(size)`
  - `Not affected by turtle.up() or down()`

Decision

```
if a >= b:
    print()
```

```
if a >= b:
    print()
elif b>=c:
    print()
else:
    print()
```

```
Common operators:
>=, <=, >, <, ==, !=
and, not, or
```

Be careful!

- **indentation** is critical!
- **Colon** is necessary!
- if can be **nested**!

```
if 5%3:
    do_something if 5%3 != 0
```

Loops: while and for

```
while a < b:
    do_something()
```

Do something as long as a<b is true

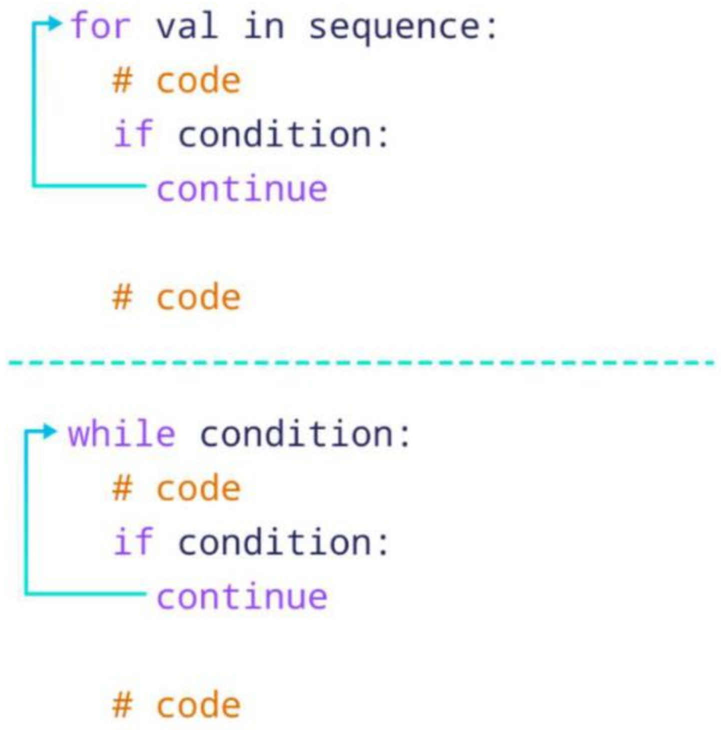
```
for item in list:
    do_something()
```

Do something for each item

```
for item in range(1,4):
    do_something()
```

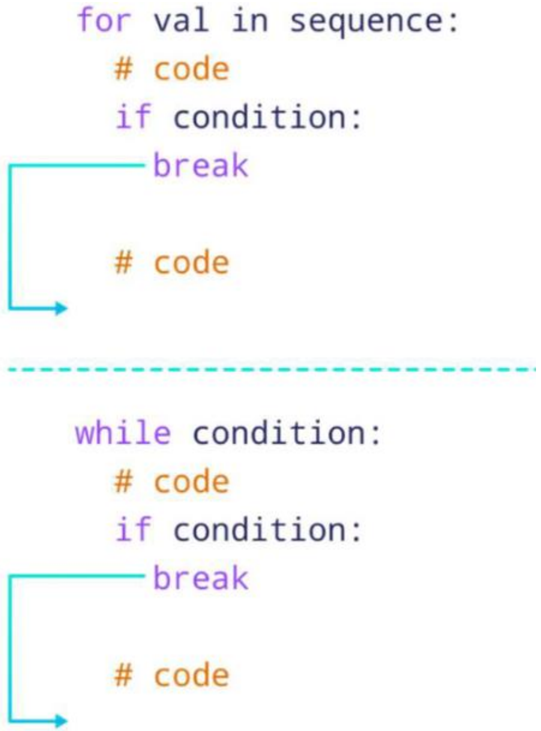
Loops -- control loops

In nested loop, break/continue only works on the loop where they are



```
for val in sequence:
    # code
    if condition:
        continue

    # code
```

```
while condition:
    # code
    if condition:
        continue

    # code
```



```
for val in sequence:
    # code
    if condition:
        break

    # code
```

```
while condition:
    # code
    if condition:
        break

    # code
```

`continue`: skip current iteration and start the next iteration.

`break`: stop entire loop and jump out of the loop `end` **will never be generated!**

`range(start, end, step)`

- `range(1, 6)`
  - `1,2,3,4,5`
- `range(1, 6, 2)`
  - `1,3,5`
- `range(6, 1, -1)`
  - `6,5,4,3,2`

```
for x in range(1, 6)
    print(x)
```

```
for _ in range(1, 6)
    print()
```

- `range(6)`
  - **`0`**`,1,2,3,4,5`
- `list(range(0)) is []`
  - `Empty list`

`list(range(start,end,step))` **will generate a list**

```
print(list(range(1,6)))
=> [1,2,3,4,5]
print(range(1,6)) =>
range(1,6)
```

List, tuple, string

- list_friends = ["Chan", "May", "Peter"]
  - list_friends[0]: "Chan"
  - len(list_friends): 3
- tuple_friends = ("Chan", "May", "Peter")
  - tuple_friends[0]: "Chan"
  - len(tuple _friends): 3
- string_friend = "chan"
  - string_friend[0]: "C‶
  - len(string_friend): 4

List, tuples, strings – common functions

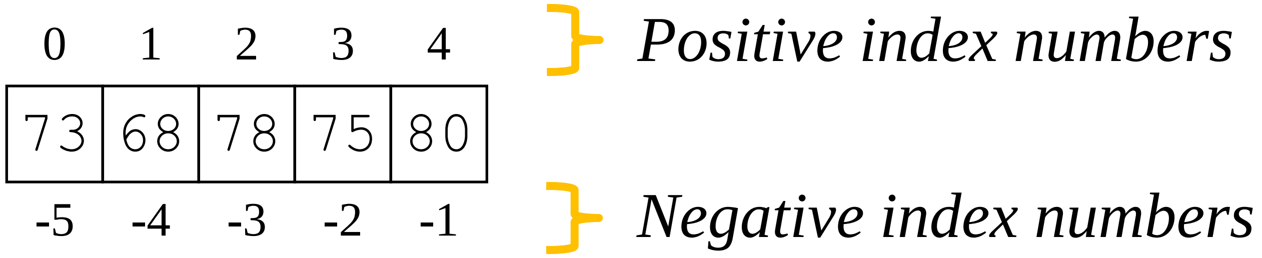
- `len()`: the number of elements in list
- `insert(index, x)`: insert x at index
- `remove(x)`: remove the first element that is equal to x
- `count(x)`: sort how many x in list
- `index(x)`: the index of the first element that is equal to x
- `append(x)`: add something after the last one
- `sort()`: sort elements in list (from small to large)
- `reverse()`: reverses the elements of list
  - `words.reverse() or words.sort()`
  - not `words = words.reverse()`
  - not `words = words.sort()`

```
A + B : add two sequences
A * int: repeat A for int times
```

```
["Chan", "Mary"] + ["May", "Wong"] = ["Chan", "Mary", "May", "Wong"]

["left", "right"] * 2 = ["left", "right", "left", "right"]
```

```
info = [21, 19, 18, 25, 20, 26]
print(info[1:3])  =>     [19, 18]
```

```
x = [ 73, 68, 78, 75, 80 ]
```



0	1	2	3	4
73	68	78	75	80
-5	-4	-3	-2	-1

Positive index numbers

Negative index numbers

```
things =  [ [ [1, 2],  [3, 4]   ],
            [ [5, 6],  [7, 8]   ],
            [ [9, 10], [11, 12] ] ]

print( things[1][0][1] ) → 6
```

```
len(things)=3
len(things[0])=2
```

List, tuples, strings – indexing

```
     0  1  2  3  4    Positive index numbers
x = [A, B, C, D, E]
    -5 -4 -3 -2 -1    Negative index numbers
```

```
x[3]-> [D]

x[0]-> [A]

x[-1]-> [E]
```

List, tuples, strings – Slicing

mydata[*start_index* : *target_index* : step]

```
        0   1   2   3   4    Positive index numbers
x = [A, B, C, D, E]
       -5  -4  -3  -2  -1    Negative index numbers
```

```
x[:3]-> [A,B,C]              x[4:0:-1]-> [E,D,C,B]
                             x[4::-1]-> [E,D,C,B,A]
x[0:5:2]-> [A,C]             x[::-1]-> [E,D,C,B,A]
x[3:]-> [D,E]                x[4:-:-1]-> []
```

```
samples[ ::3]-> keep every third one (skip two of them)
samples[ :int(len(samples)*.25)]
```

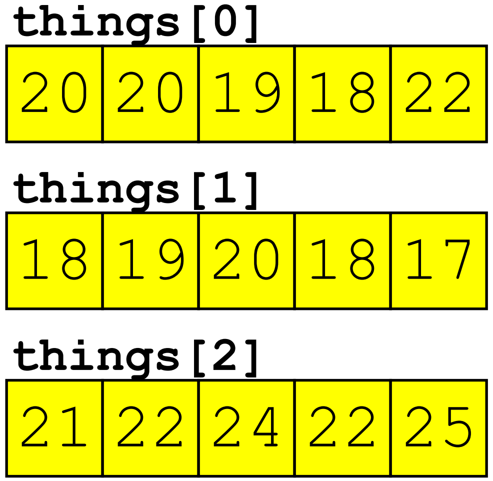
Slicing – change data (only for list!)

```
info = [21, 19, 18, 21, 20, 19]

info[1:3] = [25, 27]

print(info)    [21, 25, 27, 21, 20, 19]
```

N-D sequences

```
Two (( means that this is a 2-D tuple
Three [[[ means that this is a 3-D list
```

```
things = ((20, 20, 19, 18, 22),
          (18, 19, 20, 18, 17),
          (21, 22, 24, 22, 25))
```



things[0]

20	20	19	18	22

things[1]

18	19	20	18	17

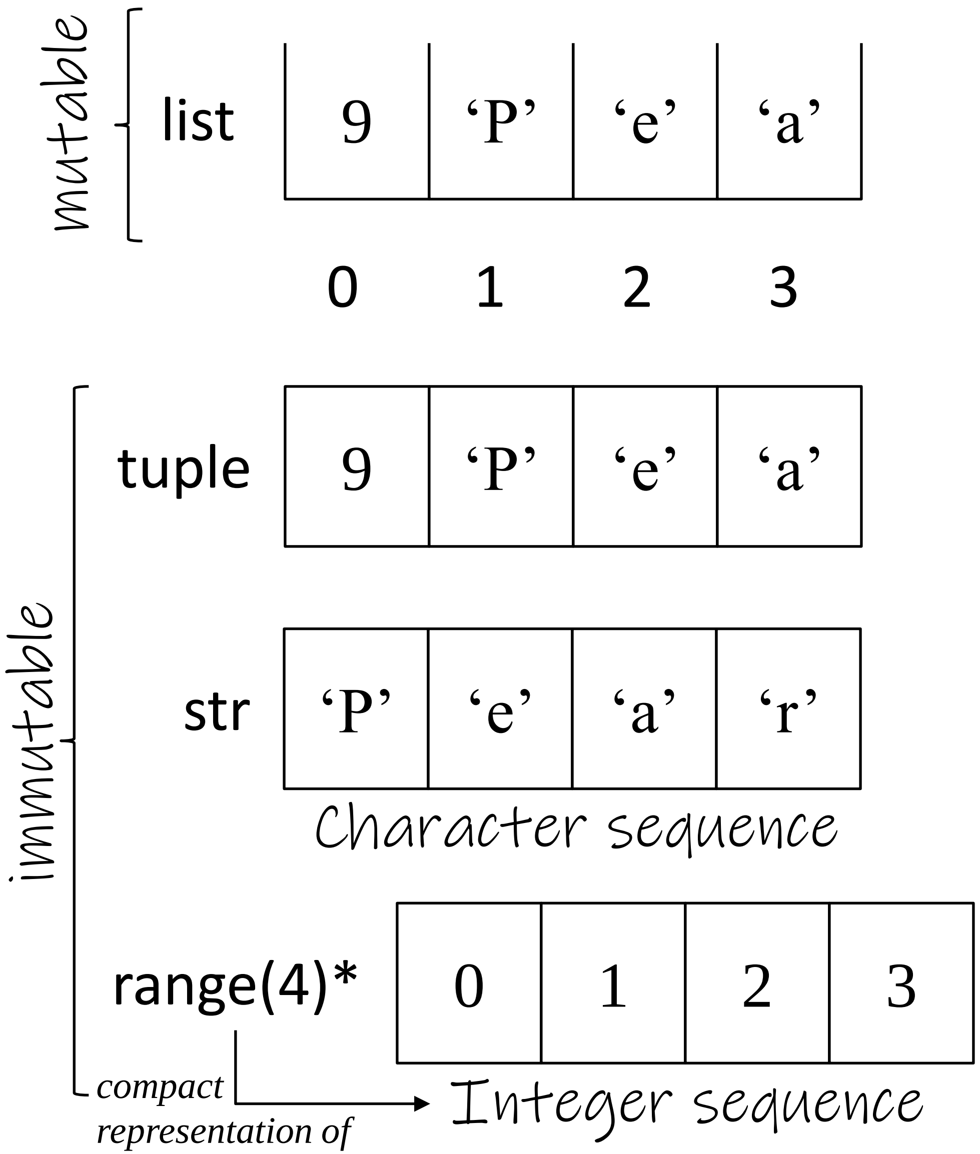
things[2]

21	22	24	22	25

```
print(things[2][1])  ➡  22

print(things[0])     ➡  (20, 20, 19, 18, 22)
```

```
len(things) = 3
len(things[0]) = 5
```

- `len()` doesn’t count inside the lists which are inside the list



Mutable	[], for, len, count, index, insert, remove, append, reverse, sort, extend
Immutable	[], for, len, count, index

```
a = [9,‘P’,‘e’,‘a’]
    (9,‘P’,‘e’,‘a’)
    "Pear"
    range(4)
print(a[3])

         [9,‘P’,‘e’,‘a’]
         (9,‘P’,‘e’,‘a’)
for e in "Pear"
         range(4)
  print(e)
```

Functions

function name

```
def show_response( name ):
    if name == "Dave":
        print("What a good name!")
    else:
        print("How are you?")
    return name, 1
```

variable name, can be multiple variables

```
name, x = show_response(name)
```

Function must be used after it is defined

Functions – local and global variables

```
Values = [1, 10, 100]
def f1():
    local_var_one = "Hello"
    return local_var_one

def f2():
    local_var_two = "Greetings"
    return local_var_two

print(f1())
print(f2())
print(local_var_one)
print(local_var_two)
```

We can only use local_var_one in this area

Local variable

We can only use **local_var_two** in this range

✗ NameError: name 'local_var_one' is not defined

We can use **Values** anywhere

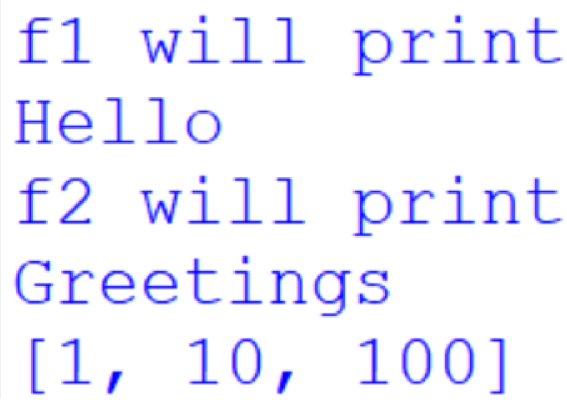
Global variable

Functions – local and global variables

```
var = [1, 10, 100]
def f1():
    var = "Hello"
    print(var)
def f2():
    var = "Greetings"
    print(var)
```

If a local variable and a global variable have the same name, priority is given to the local variable

```
print("f1 will print")
f1()
print("f2 will print")
f2()
print(var)
```



```
f1 will print
Hello
f2 will print
Greetings
[1, 10, 100]
```

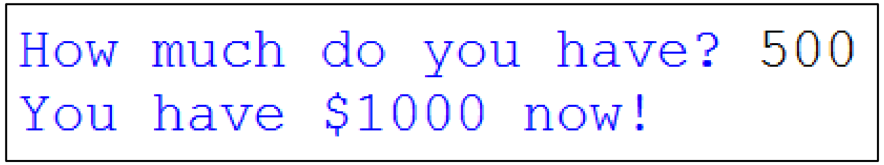
Change local variable will not affect global variables

Functions – local and global variables

```
def magic_trick():
    global money

    if money < 1000:
        money = money + 500

money = int(input("How much do you have? "))
magic_trick()
print("You have $" + str(money) + " now!")
```

We tell Python that when we refer to `money` *in the function, it means the global variable* `money`

This line changes the value of the global variable



```
How much do you have? 500
You have $1000 now!
```

Recursive function

- A recursive function calls itself

```
def Fibonacci(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return Fibonacci(n-1)+Fibonacci(n-2)
```

Base case: The base case is the simplest scenario that does not require further recursion. **Termination** of the recursion

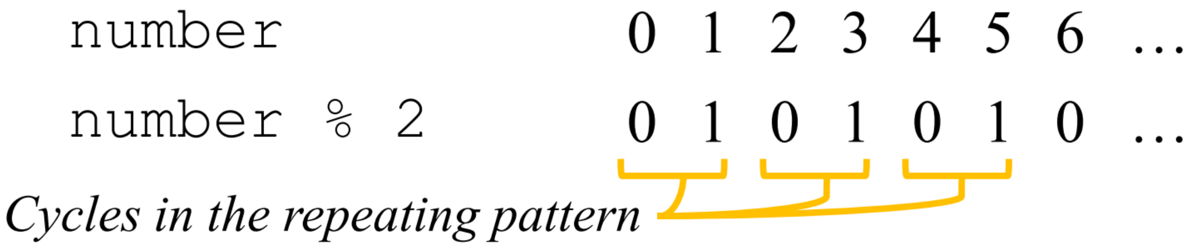
recursive case: calls itself with the modified arguments.

```
Fibonacci(1) is 1
Fibonacci(2) is 1 = F(1)+F(0)
Fibonacci(3) is 2 = F(2)+F(1)
Fibonacci(4) is 3 = F(3)+F(2)
Fibonacci(5) is 5 = F(4)+F(3)
Fibonacci(6) is 8 = F(4)+F(3)
```

Recursive function usually can be converted to iterative code (using for/while loops)

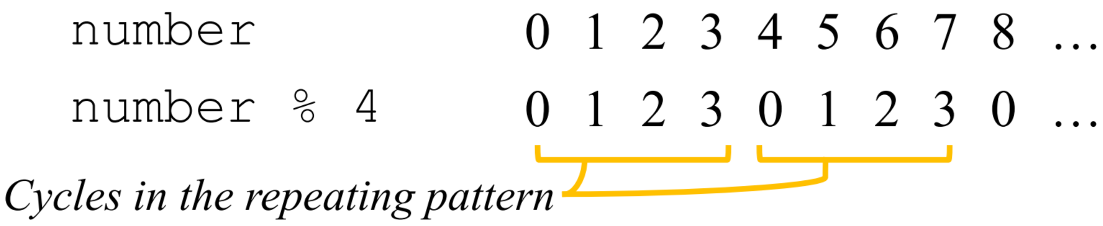
Numbers – remainder

- A % B: the remainder after division
- 10 % 2 = 0
- 10 % 3 = 1
- Remainder is useful for controlling repeated patterns



`number`	0	1	2	3	4	5	6	…
`number % 2`	0	1	0	1	0	1	0	…

Cycles in the repeating pattern



`number`	0	1	2	3	4	5	6	7	8	…
`number % 4`	0	1	2	3	0	1	2	3	0	…

Cycles in the repeating pattern

Numbers – int and float

- `int(1.9) = 1`
  - `always discard the number after decimal place`
- `int("1") -> 1`
- `int("right") -> error`

- `float(1) = 1.0`

- `round(0.5) -> 0, round(1.5) -> 2`
  - `For x.5 -> round to the nearest even int`
- `round(0.4) -> 0, round(1.4) -> 1, round(1.9) -> 2`
  - `Other wise, round to the nearest int`

Types

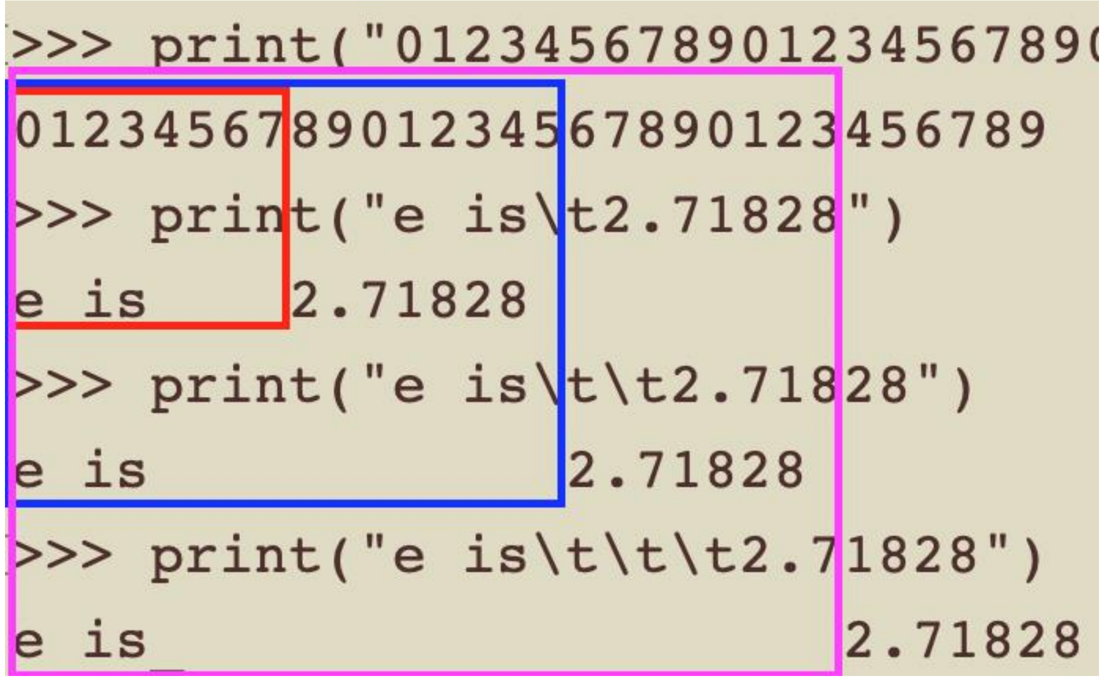
- `type(1) -> int`
- `type(1.0) -> float`
- `type("1") -> string`
- `type(["1"]) -> list`

Special characters

- `"\t"` is a tab. It is not a fixed length
  - It looks like a sequence of spaces but it is not!
  - When we **press** tab in keyboard, it will be 4 whitespaces
  - When python **prints** \t, it will move the cursor to n-th column, such that n % 8 is 0
    - In other words, it looks like we “pad” space until the length of string % 8 is equal to 0
- `"\n"` means a “new line”
- `string.rstrip()` removes any space, \n and \t at the **right** side of string.
- `string.split("\t")` splits string using `"\t"`

\t

```
print("01234567890123456789")
print("e is\t2.71828")
print("e is\t\t2.71828")
print("e is\t\t\t2.71828")
```



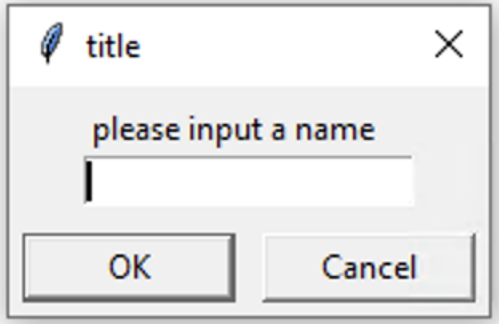
```
>>> print("012345678901234567890
01234567890123456789
>>> print("e is\t2.71828")
e is    2.71828
>>> print("e is\t\t2.71828")
e is            2.71828
>>> print("e is\t\t\t2.71828")
e is                    2.71828
```

Turtle - Input box

```
var=turtle.textinput("title","please input a name")
```

A prompt will be displayed and ask for inputs.

Inputs typed by users will be saved to `var`



title

please input a name

File IO

```
#Open the file for writing text
myfile = open(filename, "wt")

one_line = "abcd + "\n"

# Save the string to the file
# write will not automatically pad \n
myfile.write(one_line)

# Close the file
myfile.close()
```

```
# Open the file for reading
myfile = open(filename, "r")

for line in myfile:
    # Handle each line, one by one
    # print(myfile) does not work!

    # Remove the end-of-line
    line = line.rstrip()
    # do something here

myfile.close()  # We have finished,
now close the file
```

RGB Colors

- Each value of R/G/B needs to be [0, 255]
- White is R255, G255, B255; Black is 0, 0, 0
- `turtle.colormode(255)`
- `turtle.bgcolor(int(red), int(green), int(blue))`
  - `Must be int!`

min() and max()

- we use max() to make sure the value doesn’t go **below** zero
  - `value = max(value, 0)`
- we use min() to make sure the value doesn’t go **beyond** 255
  - `value = min(value, 255)`

Operators

- `A**B` **->** `A^B`
- `A//B` -> integer division
- `Any number != 0` **->** `True;`
- non-empty list/tuple/string **->** `True`
- `count += 1` is the same as `count = count + 1`
- ele `in` list -> check if ele is one element of list

Increasing precedence

- Highest precedence -

`( )`

`**`

`-x`, `+x`

`*`, `/`, `%`, `//`

`+`, `-`

`<`, `>`, `<=`, `>=`, `!=`, `==`

`in`, `not in`

logical `not`

logical `and`

logical `or`

- Lowest precedence -

Dict

```
# Create a dict

heads = {"David":  (589, 106, 48, 63),
         "Gibson": (474, 102, 44, 58),
         "Paul": (522, 162, 55, 68)
}

# get

Value = heads ["David"]

# update

heads ["David"] = (1,2,3,4)


# delete a key (and its value)

del heads ["David"]
```

Almost anything can be used as a key, but not **List.**
List can be used as **value**.

```
# go through via keys

for key in heads.keys():

    print(key) # David, Gibson, Paul
```

```
# go through via values

for value in heads.values():

    print(value)

#(589, 106, 48, 63)

#(474, 102, 44, 58)
#(522, 162, 55, 68)
```

```
# go through via key+value pairs

for key,value in heads.items():

    print(key,value)

# David, (589, 106, 48, 63)

…
```

Class and Object

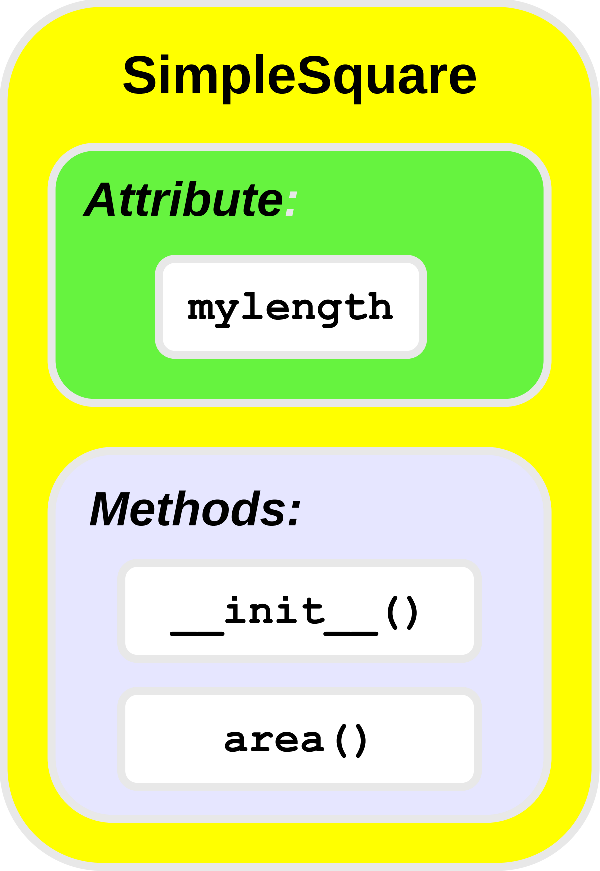
```
class SimpleSquare:
    def __init__(self, length):
        # this constructor is invoked whenever
        # mySimpleSquare = SimpleSquare(len)
        self.mylength = length

    def area(self):
        return self.mylength * self.mylength
```

The first parameter of a function in a class always has to be `self` , which means **itself**/**myself** (meaning the instance of the class)

We do **not** pass self to methods when we call it!

To access **attributes**: mySimpleSquare.mylength

To invoke **methods**: mySimpleSquare.area() -> **no self**



SimpleSquare

Attribute:

mylength

Methods:

__init__()

area()

Common mistakes

```
list(range(0)) is []
No error!
```

```
list(range(2)) is [0,1]
Start from 0
```

```
if 5%3:
    do_something if 5%3 is not 0
```

Function must be used after it is defined

We cannot change things in tuple and string!

Square brackets and parentheses must be paired

Common mistakes

Square brackets and parentheses must be paired

```
list(range(0))
list(range(0) is wrong!
```

" " is a string with space, len(" ") is 1
"" is an empty string, len("") is 0

Tips

- Use `turtle.speed()` to save your time in execution
  - Faster speed: quickly see the results
  - Slower speed: check the steps
- Use `turtle.hideturtle()` and `showturtle()` smartly
  - To show the current orientation of turtle!

- ALWAYS read the questions carefully!
- ALWAYS understand what is asked for you to input!
  - a full command? A number?
  - capital letter or not?
- Validate your code using the examples