

Introduction to Python

Fundamental input/output, datatypes, arithmetic expression

01204111 Section 1

Formal and Natural languages

Natural languages

- The languages people speak
- Have to figure out what the structure of the sentence is, this process is called **parsing**
- Full of Ambiguous
- Lots of Redundancy
- English, Spanish, Thai

Formal languages

- Designed by people for specific applications
- Strict rules about syntax
- Nearly unambiguous
- Less redundant
- Chemical structure : H_2O
Math Equation : $3+3=6$

Programming Language

Syntax - Semantics

- **Syntax** : Structure of a program.
- **Semantics** : The meaning of program.

I love programming

Correct Syntax and Semantics

love I programming

Programming love I

I programming love

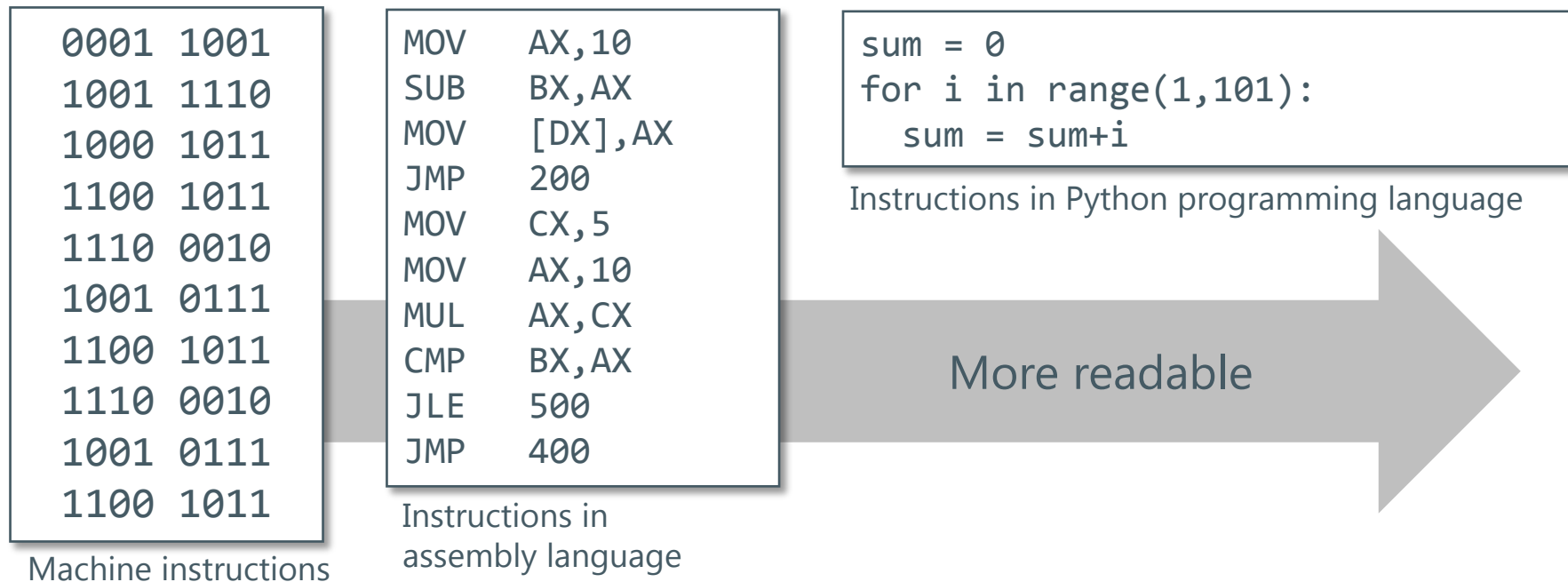
Wrong Syntax
Wrong Semantics

Programming loves me

Correct Syntax
Wrong Semantics

A computer program

- A **computer program** is a sequence of **instructions** to be executed by computers.
- Examples of computer programs in various forms:



Inside the memory

- The smallest unit of information that can be processed by digital computers is a single binary digit (a **bit**), which can be either 0 or 1.
- We usually group them in groups of 8 bits, each called a **Byte**.
- A lot of bytes can be stored in a memory unit.
 - 1 kB = 1,000 bytes
 - 1 MB = 1,000,000 bytes
 - 1 GB = 1,000,000,000 bytes

0 1

Two bits

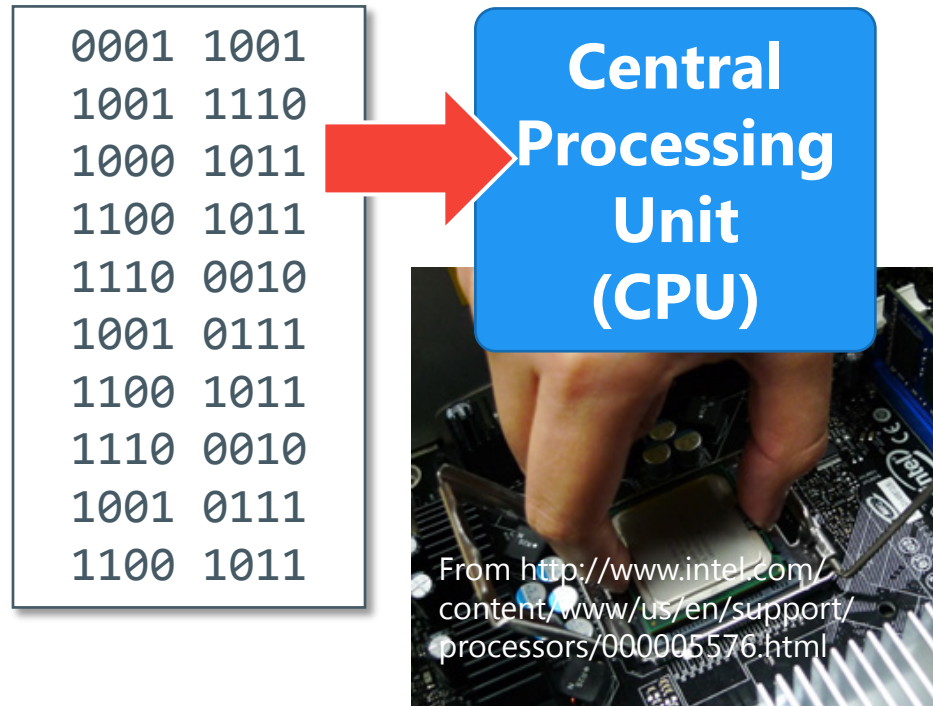
0 1 0 1 1 1 0 0

One byte

IEEE-1541 recommendation

- 1 kiB = 2^{10} bytes = 1,024 bytes
- 1 MiB = 2^{20} bytes = 1,024 kiB
- 1 GiB = 2^{30} bytes = 1,024 MiB

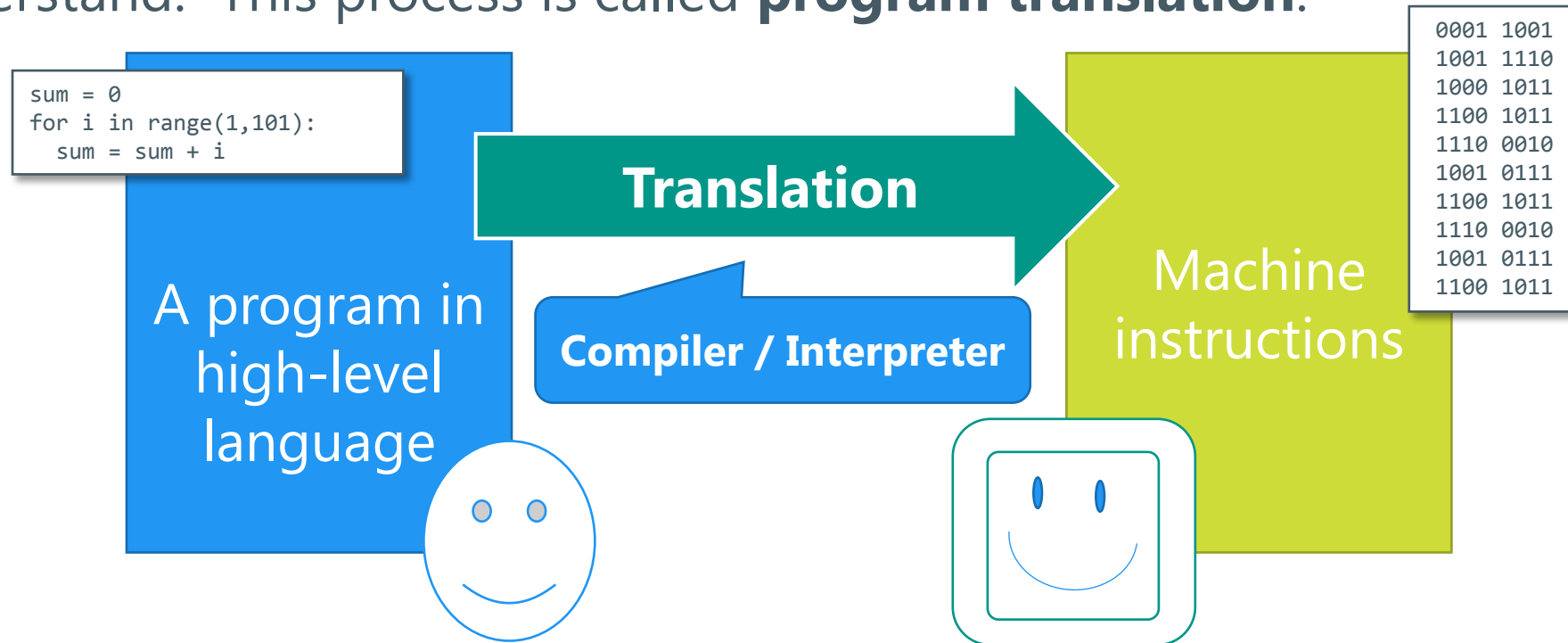
The instructions



- The memory, not only keeps the data to be processed with the CPU, but it also keeps the instructions.
- These instructions are in the format that the **CPU** can easily understand, referred to as "**machine instructions.**"
- When writing a program, we rarely write in machine instructions.

From programs to instructions

- Instead of working directly with machine instructions, people usually develop software with higher-level programming languages.
- But the program must be translated into a form that the computer can understand. This process is called **program translation**.

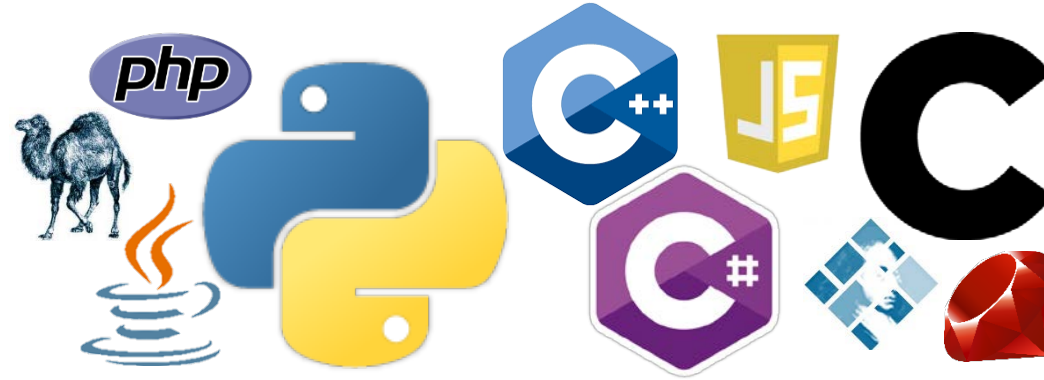
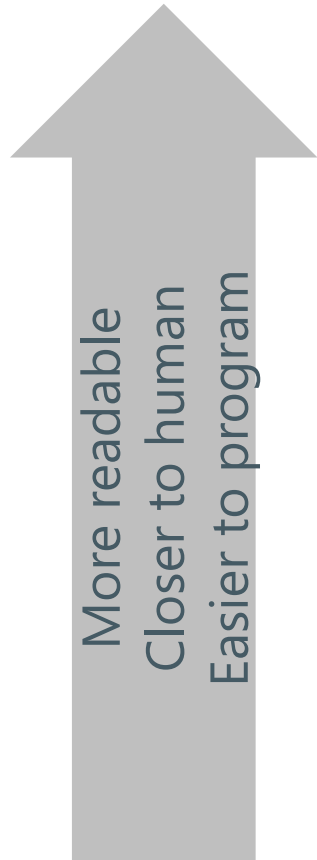


Why do we need Computer Language?

- **Problem Solving** : The Ability to formulate problems, think creatively about solutions, and express a solution clearly and accurately.



Programming Language



High-Level Language

Portable : can run on different kinds of computer (with few or no modifications)

MOV	AX,10	SUB	BX,AX
MOV	[DX],AX	JMP	200
MOV	CX,5	MOV	AX,10
MUL	AX,CX	CMP	BX,AX

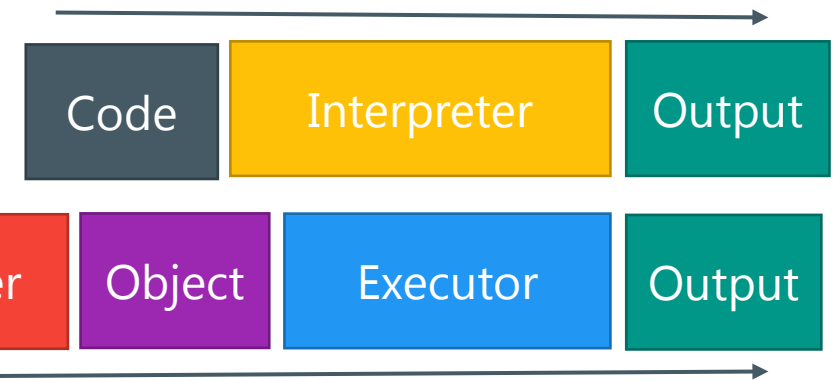
Low-Level Language

Assembly Language

0001	1001	1001	1110	1000	1011	1100	1011
1110	0010	1001	0111	1100	1011	1110	0010

Machine Code

Translation



- Interpreter

- Translates program one statement at a time
- Less time to analyze but overall execution time is slower
- Continues translating the program until meet the first error

- Compiler

- Scan the entire program and translates it as a whole into machine language
- Take large time to analyze the source code but execution time is faster
- Generates the error message only after scanning the whole program

What is Python

- Python is a programming language coming with simple syntax and a lot of tools

Hello CPE students.



History of Python

- Guido van Rossum creates Python when he was a Google employee.
- His first intention is to create a programming language that is
 - equipped with tools to help in doing stuffs
 - **as easy to read as plain English**



Python is good, Python is bad.

Python is good

- Equipped with tools and libraries to help in doing stuffs
- Plain and easy-to-understand syntax, as clear as plain English
- Python expands its field very far: from web framework to machine learning.

Python is bad

- Lacks of fundamental concepts like datatypes
- Dynamic typed language's got both pros and cons
- Interpreter-based language means Python performance is much lower than compiler-based (like C)

Python coding mode

- **Interactive mode**

```
# python  
>>> print("Hello World")  
Hello World
```

The chevron, >>>, is the prompt the interpreter use to indicate that it is ready. If you type 1+1, the interpreter replies 2.

- **Script mode**

```
# echo print("Hello World") > hello.py  
# python hello.py  
Hello World
```



```
print("Hello World")
```

hello.py

Your first Python program

- Try writing the code below and running it

```
print("Hello World")
```

Output

```
Hello World
```

- The command is understandable, and its task could be guessed, even for those without programming knowledge.

Printing

- **print** function is for printing texts to the screen

```
print("Hello World")
```

Output

```
Hello World
```

- Every print function will make a new line after its execution

```
print("Hello CPE students")  
print("Welcome to 111")
```

Output

```
Hello CPE students  
Welcome to 111
```


Printing

- One print command may accept many strings

```
print("same", "line")
```

Output

```
same line
```

- Separated with comma, texts will be printed in the same line and separated with space.

Printing

- This is how you print texts to one single line with multiple print commands:

```
print("Kasetsart", end=" ")  
print("University")
```

Output

```
Kasetsart University
```

- You can use **end** to tell print function that what character will be used instead of new line (new line is a character too)

Escape sequence

- Some character can cause problems in programs. For example,

```
'I'm Engineer'
```

- Python thinks that single quote mark in '**I'm**' ends string, you can fix it by adding backslash (\) before quote.

```
'I\'m Engineer'
```

- Characters with backslash are called **escape sequence**.

Escape sequence

- These are escape sequence which you may usually use

Escape sequence	Meaning
\\	Backslash (\)
\'	Single quote
\"	Double quote
\n	New line
\t	Tab space

Exercise

- Write a program to introduce yourself by using this pattern
- Only one print function is allowed

```
tab space My name is your name.  
tab space I'm your age years old.  
tab space Nice to meet you, CPE friends  
  
tab space \/\/\/\ "Lorem ipsum dolor sit amet" /\^V
```

Variable

- Variable is used for storing value by using equal sign (=)
- The **value** on the right side of equal sign is stored in variable(**identifier**) on the left side

$$x = 10$$

- Variable must named by beginning with letter (A-Z and a-z) or underscore (_)
- Python has **Reserved Words** which can't be used for naming variable.

Reserved Words

and	del	from	not
as	elif	global	or
assert	else	if	pass
break	except	import	print
class	exec	in	raise
continue	finally	is	return
def	for	lambda	try
while	with	yield	

Data type

- In some programming language, you have to tell it that what kind of value will be stored in variable (such as C language which uses **int** for integer, **char** for character, etc.)
- But Python can choose appropriate type of variable for the value which you provided



Leave it to me.
I will make your
programming life
easier.

Data type

- Python has 5 standard data type

1. Numbers

- **Integer** (e.g. 1, 15, 500)
- **Floating point** (e.g. 3.14, 24.7)

2. **String** : a character or a set of characters represented in *single quote* or *double quote* mark (e.g. "CPE", 'Hello World')

```
myNumber = 10  
myString = "Hello CPE students!"
```

Data type

- 3. **List** : set of items ,which can be different type, separated with comma (,) and enclosed within square brackets **[]** (e.g. [1, 'a', 1.5])
- 4. **Tuple** : similar to list but it's items cannot be edited. it's values are enclosed within parentheses **()**
- 5. **Dictionary** : similar to list but using key to indicate value. it's values are enclosed within brackets **{ }**

```
myList = [1, "I am list", 555]  
myTuple = (2, "I am Tuple", 666)  
myDict = {name: "John", age: 18}
```

Type function

If you are not sure what type a value has, you can use `type()` to inspect them

```
print(type('Hello World'))  
print(type(17))  
Print(type(3.2))
```

Output

```
<type 'str'>  
<type 'int'>  
<type 'float'>
```

Printing

- You can print variable's value or constant to screen by using print function like the code below

```
a = 5  
print(a)  
print(6)
```

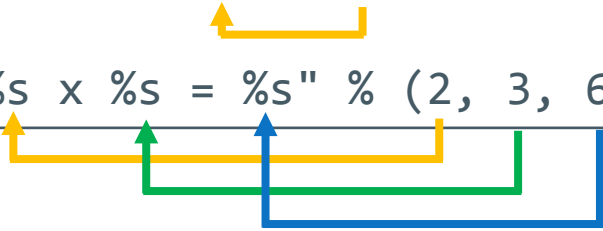
Output

```
5  
6
```

String formatting

- There are 3 ways to combine string with variable or constant
- The first way is using string modulo operator (%)

```
a = 5  
print("value = %s" % a )  
print("%s x %s = %s" % (2, 3, 6))
```



Output

```
value = 5  
2 x 3 = 6
```

String formatting

`print ("Art: %5d, Price per Unit: %8.2f" % (453, 59.058))`

The diagram illustrates the components of the string formatting code. A yellow dashed line labeled 'placeholder' points to the format specifiers `%5d` and `%8.2f` within the string. A green bracket labeled 'Format String' spans the entire string `"Art: %5d, Price per Unit: %8.2f"`. An arrow labeled 'String Modulo Operator' points to the `%` symbol. Another green bracket labeled 'Tuple with values' spans the tuple `(453, 59.058)`.

- Python will replace placeholders with items in tuple after string modulo operator respectively

http://www.python-course.eu/python3_formatted_output.php

String formatting

- Placeholder is consist of 3 parts

%A.BC

- A : Number
 - If A is begin with zero (0), value will be prepended with zero when the value length is smaller than A
 - Otherwise, value will be prepended with space instead of zero
- B : Number
 - If data type is floating point number, B is precision of value
 - Otherwise, it is length of value
- C : Character
 - Telling the type of value

String formatting

- Value types which you may usually use

Placeholder	Meaning
%d	Integer
%f	Floating point
%s	String (but every data type can use %s)

String formatting

Example

```
pi = 3.14159
print("%s" % pi)
print("%.2s" % pi)
print("%d" % pi)
print("%4d" % pi)
print("%04d" % pi)
print("%.4d" % pi)
print("%f" % pi)
print("%.4f" % pi)
```

Output

```
3.14159
3.
3
    3
0003
0003
3.141590
3.1416
```

String formatting

- Second way is adding **f** before string which you want to format

```
a = 5  
b = 6  
print(f"value1 = {a}, value2 = {b}")
```

Output

```
value1 = 5, value2 = 6
```

String formatting

- Third way is adding `.format()` after string that you want to format

```
print("pi = {}".format(3.14))  
print("e = {}".format(e = 2.72))
```

Output

```
pi = 3.14  
e = 2.72
```

Input

- Let's know how to get input for your program

```
a = input("Enter something: ")
```

- Running the code above, program will print "Enter something: " to your screen and wait you to enter input.
- The value which you just entered will be stored as **string** type in variable **a**.

Input

- If you want to change data type of input, you can do it by **casting**

```
a = int(input("Enter something: "))
```

convert input to integer type

- **Casting** is one of the way to convert data type by telling the type which you want such as
 - **int(...)** : convert to integer
 - **float(...)** : convert to floating point
 - **str(...)** : convert to string
 - etc.

Bugs

- **Bugs** is programming errors. The process of tracking them down is call **debugging**.

- **Syntax Errors** : cased from wrong structure of program

Ex.

print "Hello World)



print ("Hello World")



Bugs

- **Runtime errors** : appear after the program has started running. This type Also called **exceptions** because they usually indicate that something exceptional has happened.

Ex.

```
a = [0,1,2,3]  
print(a[4])
```



Indexed out of range

Bugs

- **Sematic errors** : the program run successfully and the computer will not generate any error messages but it will not do the right thing.

Ex. Find the average of 45, 89, 63

```
av = 45 + 89 + 63 / 3  
print(av)
```



```
av = (45 + 89 + 63) / 3  
print(av)
```



A Typical Python program

Flow of
program



```
"""Rectangle area calculator
```

```
Ask rectangle's length and width from user,  
then compute rectangle area
```

```
"""
```

```
import math
```

```
def rectangle_area(length,width):
```

```
    """Compute area of rectangle of specified length and width"""
```

```
    return length*width
```

```
# ask user to input length and width of a rectangle
```

```
length = int(input("Enter length: "))
```

```
width = int(input("Enter width: "))
```

```
# then compute and print out the area
```

```
area = rectangle_area(length,width)
```

```
print(f"Rectangle area is {area}")
```

Docstrings

Function
definition

Comments

Comments & Docstrings

- They are in the code to provide insights into what code is doing, how code works, or give other notes
 - They do not affect how program works
- Docstrings are displayed when calling for help

```
def rectangle_area(length,width):  
    """Compute area of rectangle of specified length and width"""  
    return length*width
```

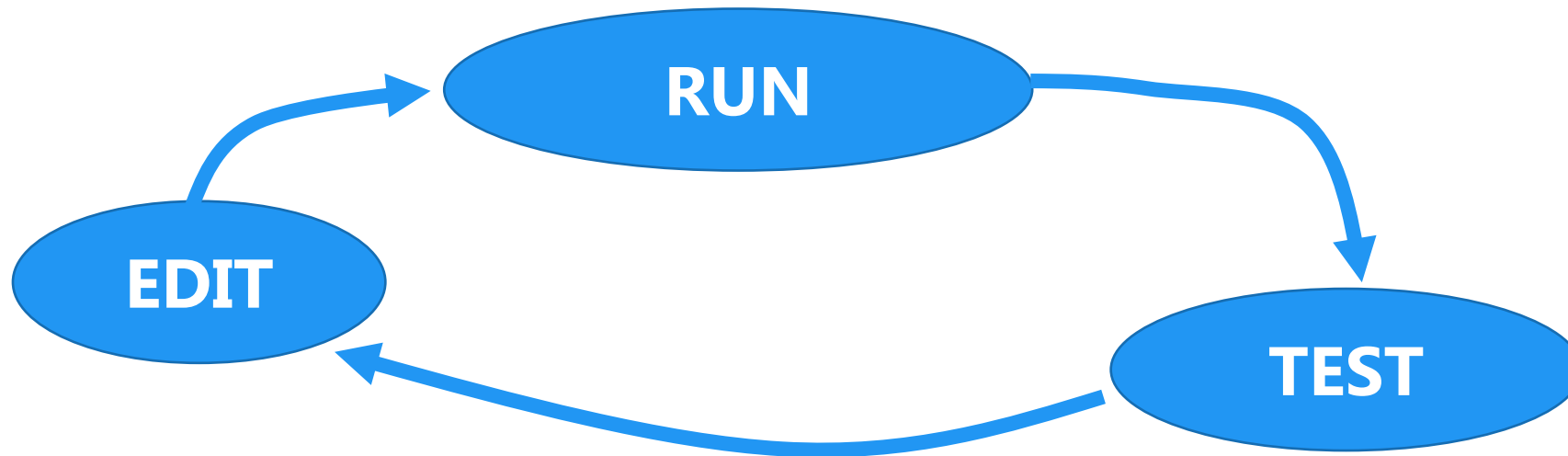
docstring

```
# ask user to input length and width of a rectangle  
length = int(input("Enter length: "))  
width = int(input("Enter width: "))
```

comment

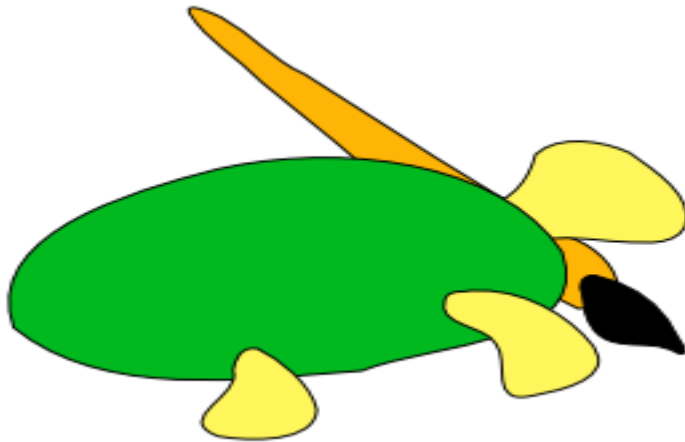
Edit-Run-Test Loop

- Because everything in life doesn't always work the first time you try it, your program may not always do exactly like what you want.
- It may be correct in some case, but it might fail in some other. Therefore, you need to **test** your program. If it is not correct, you have to **fix** it (**debug** it), and try to test it again.
- Your process for writing a Python program would look like this.

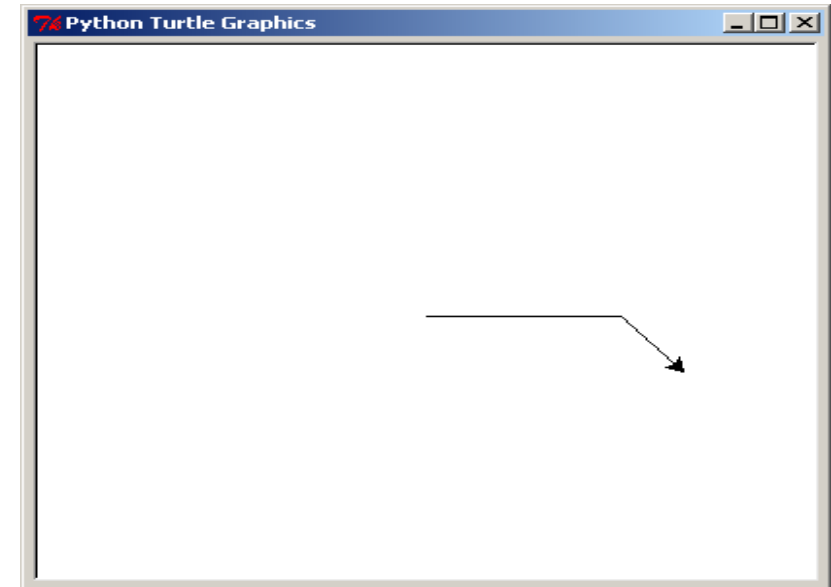


Python Turtle Graphics

- Python comes with Turtle graphics module that allows you to control movement of a robotic turtle on screen
- The turtle carries a pen and draws over the path he moves on



Graphics illustrated by Jittat Fakcharoenphol



Basic Turtle Movements

- `turtle.forward(d)` – tell Turtle to walk forward *d* steps
- `turtle.right(a)` – tell Turtle to turn right for *a* degrees
- `turtle.left(a)` – tell Turtle to turn left for *a* degrees

Tell Turtle to draw something

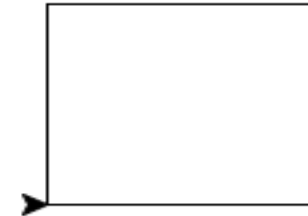
- The code is executed *sequentially* from top to bottom
- What is Turtle drawing?

Flow of
program



```
import turtle

turtle.forward(100)
turtle.left(90)
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
turtle.left(90)
```



Abstraction with *subroutines*

- A set of instructions can be defined into a subroutine so that they can be called for execution later

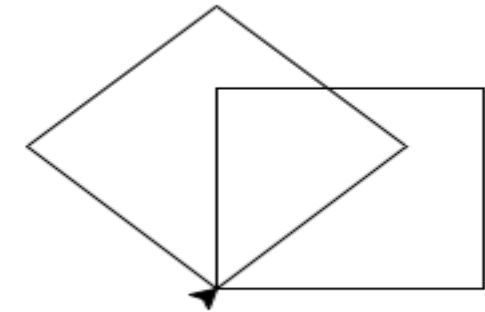
Define a subroutine
"draw_square"

```
import turtle

def draw_square():
    turtle.forward(100)
    turtle.left(90)
    turtle.forward(100)
    turtle.left(90)
    turtle.forward(100)
    turtle.left(90)
    turtle.forward(100)
    turtle.left(90)
```

Call the subroutine
"draw_square"

```
draw_square()
turtle.left(45)
draw_square()
```



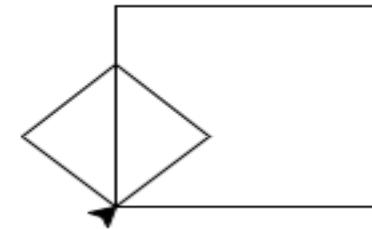
Make it more general

- The **draw_square** subroutine can be generalized to draw rectangles of any sizes

```
import turtle

def draw_square(size):
    turtle.forward(size)
    turtle.left(90)
    turtle.forward(size)
    turtle.left(90)
    turtle.forward(size)
    turtle.left(90)
    turtle.forward(size)
    turtle.left(90)
```

"draw_square"
now takes a parameter



```
draw_square(100)
turtle.left(45)
draw_square(50)
```

Draw a square of size 100

Draw a square of size 50

Get rid of *repetitive* code

- Most programming languages provide special constructs for repeated actions

```
import turtle

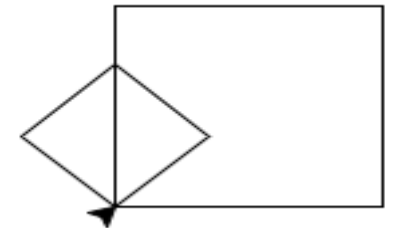
def draw_square(size):
    turtle.forward(size)
    turtle.left(90)
    turtle.forward(size)
    turtle.left(90)
    turtle.forward(size)
    turtle.left(90)
    turtle.forward(size)
    turtle.left(90)
```

```
draw_square(100)
turtle.left(45)
draw_square(50)
```

```
import turtle

def draw_square(size):
    for _ in range(4):
        turtle.forward(size)
        turtle.left(90)
```

```
draw_square(100)
turtle.left(45)
draw_square(50)
```



Execute actions *selectively*

- Some actions need to be executed only when certain conditions are met

```
import turtle
```

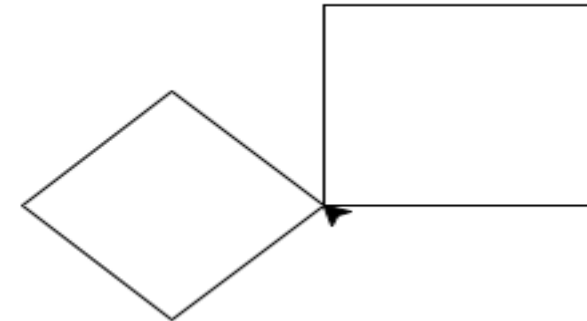
```
def draw_square(size):  
    if size > 0:
```

```
        for _ in range(4):  
            turtle.forward(size)  
            turtle.left(90)
```

```
draw_square(100)  
turtle.left(45)  
draw_square(-50)  
turtle.left(90)  
draw_square(80)
```

This code is executed only when size is greater than zero.

This call gives no result



References

- Python Slides (2017) – **Department of Computer Engineering
Kasetsart University**
- Think Python – **Allen B. Downey**