INF1002
Programming Fundamentals
Lecture 2: Python basic II

Zhang Zhengchen
zhengchen.zhang@singaporetech.edu.sg

Review

- Variables
  - Id
  - Name
  - Type and Type Casting
  - Arithmetic operators

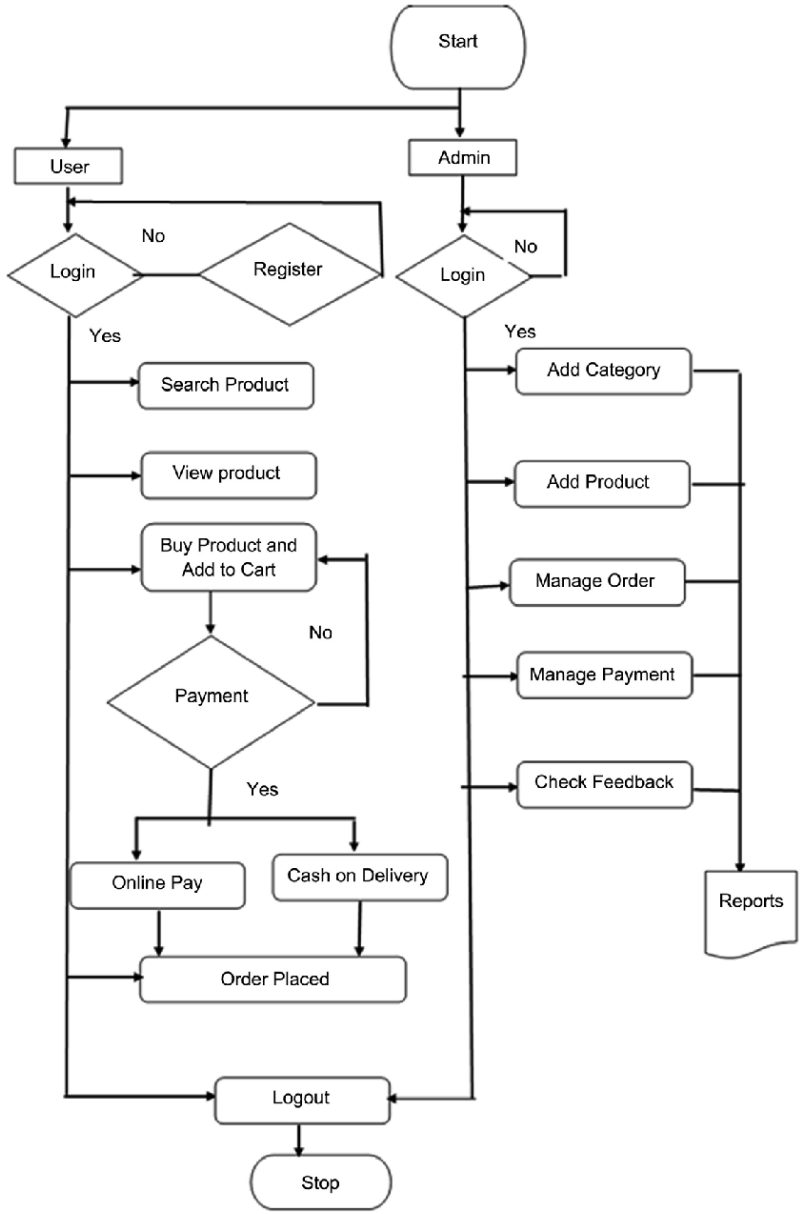
Assignment

- Assignment
  - Install the Python 3
    - Try pip install
    - Conda / Miniconda
    - Install Conda
    - Create an environment
  - Install the IDE that you like to use (i.e., Pycharm , Visual Studio Code)
    - Create a simple program
    - Try run, debug, breakpoint, check the variant value

- First lab from week 2

Outline

- Project
- **Control Flow**
  - **Conditional Control**
    - **if statement**
  - Loop Control
    - while
    - for
- Input/Output from console
- String format



https://www.researchgate.net/figure/Flow-chart-diagram-Online-e-commerce-shopping-web-app_fig1_347169962

Definition of Control Flow

- In computer science, control flow (or flow of control) is the order in which individual statements, instructions or function calls of an imperative program are executed or evaluated.

- Imperative programming
  - https://en.wikipedia.org/wiki/Imperative_programming
- Declarative programming
  - https://en.wikipedia.org/wiki/Declarative_programming

https://en.wikipedia.org/wiki/Computer_programming

Main Control Flow Structures

- **Sequential Control**
- **Conditional Control**
- **Loop Control**
- Function Call
- Exception Handling

Sequential Control

- The program executes instructions in the **order** they are written, from top to bottom.

```
a,b=10,20
print(a,b)
b,a=a,b
print(a,b)
```

```
10 20
20 10
```

Conditional Control

- We would like the program to perform different steps depending on different conditions
- Decision-making ability
  - My Five "No-Do's" at Work
    - If I know how to do it, I won't do it, because I won't learn anything new.
    - If I don't know how to do it, I won't do it, because I don't know how.
    - If it's urgent, I won't do it, because rushing can lead to mistakes.
    - If it's not urgent, I won't do it, because if it's not urgent, why should I do it now?
    - If I don't want to do it, I won't do it, because if I don't want to, how can I do it?
    - Otherwise, I will do it.

Conditional Control

- The program makes decisions based on the truth value of conditions, determining which path to take.
- Structures: **if, if-else, if-elif-else**
- Boolean variable type
  - True or False
- Boolean expression
  - An expression that evaluates to either True or False.

```python
x = 10
y = 20
print(x == y)  # False
print(x != y)  # True
print(x > y)   # False
print(x < y)   # True
print(x >= y)  # False
print(x <= y)  # True
```

```
False
True
False
True
False
True
```

Boolean Expression

- Logical Operator
  - and
  - or
  - not

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

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

Boolean Expression

- Logical Operator
  - and
  - or
  - not
- Identity Operator
  - is
    - a is b
    - id(a)==id(b)
    - a is None
  - is not

a	not a
True	False
False	True

```
mark = 90
print(f'{mark} mark>85: {mark>85}')
print(f'{mark} mark<100: {mark<100}')
print(f'{mark} mark>85 and mark<100:
{mark>85 and mark<100}')
print('==='*10)
print(f'{mark} mark<85: {mark<85}')
print(f'{mark} mark<100: {mark<100}')
print(f'{mark} mark<85 or mark<100:
{mark<85 or mark<100}')
print('==='*10)
print(f'{mark} mark>85: {mark>85}')
print(f'{mark} not mark>85: {not mark>85}')
```

True and False in Python

- Any **non-zero and non-null values** are **TRUE**
- Either **zero** or **null** is **FALSE**

```python
a = [1,2,3]
if a:
  print(f'{a} is True')
b = []
if b:
  print(f'{b} is True')
c = ''
if c:
  print(f'{c} is True')
d = 'testing'
if d:
  print(f'{d} is True')
```

```
[1, 2, 3] is True
testing is True
```

if

if expr:

statement(s)

```
drink = "Latte"
price = 0.0
if drink == "Latte":
    print("You chose Latte.")
    print("That will be $4.00.")
    price = 4.00
```

```
You chose Latte.
That will be $4.00.
```

if-else

```
if expr:
    statement(s)
else:
    statement(s)
```

```python
drink = "Tea"
price = 0.0
if drink == "Latte":
    print("You chose Latte.")
    print("That will be $4.00.")
    price = 4.00
else:
    print("Sorry, we don't have that option.")
```

```
Sorry, we don't have that option.
```

if-elif-else

```
if expr1:
    statement(s)
elif expr2:
    statement(s)
else:
    statement(s)
```

if-elif-...-else

```
if expr1:
  statement(s)
elif expr2:
  statement(s)
elif expr3:
  statement(s)
else:
  statement(s)
```

```
drink = "Latte"
price = 0.0
if drink == "Americano":
    print("You chose Americano.")
    print("That will be $3.50.")
    price = 3.50
elif drink == "Latte":
    print("You chose Latte.")
    print("That will be $4.00.")
    price = 4.00
elif drink == "Tea":
    print("You chose Tea.")
    print("That will be $2.50.")
    price = 2.50
elif drink == "Hot Chocolate":
    print("You chose Hot Chocolate.")
    print("That will be $3.00.")
    price = 3.00
else:
    print("Sorry, we don't have that option.")
```

Details of if Statement

- The Colon (:)
  - The colon is required at the end of the if and else lines.
  - It indicates that a block of code will follow.
- Indentation:
  - Python uses indentation to define the scope of the code block.
  - Consistent indentation is crucial for the correct execution of the program.
  - Typically, four spaces or one tab is used for indentation.

Nested if Statement

- Be careful of indentations

```
drink = "Latte"
size = "Large"

if drink == "Latte":
    if size == "Large":
        print("You chose a Large Latte.")
        print("That will be $5.00.")
    else:
        print("You chose a Small Latte.")
    print("That will be $4.00.")
else:
    print("Sorry, we only have Latte available.")
```

match-case Statement

```
match variable_name:
  case 'pattern 1' :
    statement1
  case 'pattern 2' :
    statement2
  ...
  case 'pattern n' :
    statement
  case _:
    print('default')
```

```
drink = "Latte"
price = 0.0
if drink == "Americano":
    print("You chose Americano.")
    print("That will be $3.50.")
    price = 3.50
elif drink == "Latte":
    print("You chose Latte.")
    print("That will be $4.00.")
    price = 4.00
elif drink == "Tea":
    print("You chose Tea.")
    print("That will be $2.50.")
    price = 2.50
elif drink == "Hot Chocolate":
    print("You chose Hot Chocolate.")
    print("That will be $3.00.")
    price = 3.00
else:
    print("Sorry, we don't have that option.")
```

In Python's match statement (introduced in Python 3.10), case _ is used as a **wildcard pattern**. It matches anything, similar to a "default" case in other languages like switch statements.

Review

- Boolean Expressions
- Logical Operators
  - and
  - or
  - not
- Identity Operators
  - is
  - is not
- if elif elif elif else
- match-case

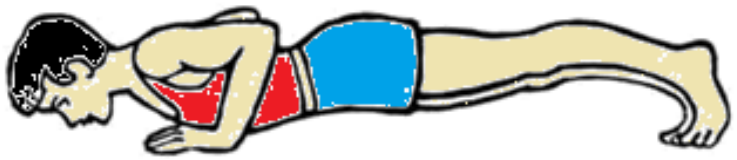
Outline

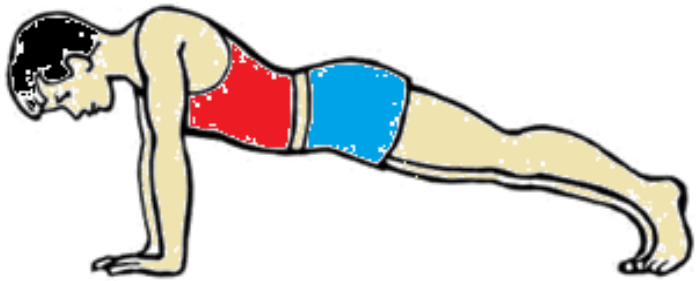
- Control Flow
  - Conditional Control
    - if statement
  - **Loop Control**
    - **while**
    - for
- Input/Output from console
- String format

https://www.researchgate.net/figure/Flow-chart-diagram-Online-e-commerce-shopping-web-app_fig1_347169962

Loop Control

- The program repeatedly executes a block of code as long as a specified condition is met.
- Structures: for, while
  - A person does 20 push-ups every day,
  - He stops once he has completed all 20 push-ups.





while Loop

- Sequential Control
  - Start
  - Push-up 1
  - Push-up 2
  - …
  - Push-up 20
  - End

- loop
  - Pseudocode:
    - pushup_count = 0
    - IF pushup_count < 20
      - Push-up
      - Increase pushup_count by 1
    - ELSE
      - Stop
    - IF pushup_count < 20
      - Push-up
      - Increase pushup_count by 1
    - ELSE
      - Stop
    - …

while Loop

The program repeatedly executes a block of code as long as a specified condition is met.

```python
def PushDown():
    pass
def PushUp():
    pass
pushup_count = 0
while pushup_count < 20:
    print(pushup_count)
    PushDown()
    PushUp()
    pushup_count += 1
print("Completed 20 push-ups!")
print(pushup_count)
```

- loop
  - Pseudocode:
    - pushup_count = 0
    - IF pushup_count < 20
      - Push-up
      - Increase pushup_count by 1
    - ELSE
      - Stop
    - IF pushup_count < 20
      - Push-up
      - Increase pushup_count by 1
    - ELSE
      - Stop
    - …

while Loop

```
while expression:
    statement1
    statement2
```

- Expression
  - Boolean Expression
  - Whether to stay within the loop
- Statements
  - Do something
  - Will change the result of the Expression

```python
def PushDown():
    pass
def PushUp():
    pass
pushup_count = 0
while pushup_count < 20:
    print(pushup_count)
    PushDown()
    PushUp()
    pushup_count += 1
print("Completed 20 push-ups!")
print(pushup_count)
```

Infinite Loops

- What happens if you forget pushup_coutnt += 1?
  - Dead Loop
  - The most common reason for a program ‘hanging’
  - To quit program execution: restart, Ctrl + C

- When Infinite Loop is useful?

  while True:

  ...

```
def PushDown():
    pass
def PushUp():
    pass
pushup_count = 0
while pushup_count < 20:
    print(pushup_count)
    PushDown()
    PushUp()
    pushup_count += 1
print("Completed 20 push-ups!")
print(pushup_count)
```

The loop may not be executed

- If the condition is False to begin with

```python
def PushDown():
    pass
def PushUp():
    pass
pushup_count = 30
while pushup_count < 20:
    print(pushup_count)
    PushDown()
    PushUp()
    pushup_count += 1
print("Completed 20 push-ups!")
print(pushup_count)
```

```
Completed 20 push-ups!
30
```

Review

- Loop
  - The program
    - repeatedly executes a block of code
    - as long as a specified condition is met.

```
while expression:
    statement1
    statement2
```

- Infinite Loops

for Loop

```
def PushDown():
    pass
def PushUp():
    pass
for pushup_count in range(20):
    PushDown()
    PushUp()
    print(pushup_count)
print("Completed 20 push-ups!")
```

for iterating_var in sequence:

statements

- Define pushup_count inside for
- Sequence
  - range(20)

for Loop

- The for loop in Python provides the ability to loop over the items of any sequence, such as a list, tuple or a string.
- It performs the same action on each item of the sequence.

- range() function
  - Generate a list of numbers
  - Generally used to iterate over with for loops
- range(stop)
  - Stop: number of integers to generate starting from 0
  - 0,1,2,3,…,19

```
def PushDown():
    pass
def PushUp():
    pass
for pushup_count in range(20):
    PushDown()
    PushUp()
    print(pushup_count)
print("Completed 20 push-ups!")
```

https://www.tutorialspoint.com/python/python_for_loops.htm

for Loop

- range(stop)
- range(start, stop)
- range(start, stop, step)
  - Start: starting num of the sequence
  - Stop: generate num up to, but not including this num
  - Step: difference between each num in the sequence

```python
x = range(10)
print(x)
print(type(x))
print(list(x))
```

```
range(0, 10)
<class 'range'>
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```python
x = range(2,7)
print(type(x))
print(list(x))
```

```
<class 'range'>
[2, 3, 4, 5, 6]
```

```python
x = range(0,10,2)
print(list(x))
```

for Loop

- The for loop in Python provides the ability to loop over the items of any sequence, such as a list, tuple or a string.

```
def PushDown():
    pass
def PushUp():
    pass
for pushup_count in range(20):
    PushDown()
    PushUp()
    print(pushup_count)
print("Completed 20 push-ups!")
```

Compare for and while

```
while boolean_expression:
    Statements
```

- Condition met or not

```
for iterating_var in sequence:
    Statements
```

- Go through the sequence
- List, tuple, string, range
- Iterable:
  - Whenever we say an iterable in Python, it means a sequence data type (for example, a list).

for Loop

- The for loop in Python provides the ability to loop over the items of **any** sequence, such as a list, tuple or a string.
- String as a sequence

```
name = 'taylor swift'
for char in name:
    print(char)
```

```
t
a
y
l
o
r

s
w
i
f
t
```

for Loop

```
for iterating_var in sequence:
    statements
```

- You may use the iterating_var in the statements
- Scenario:
  - John starts walking from his house.
  - He walks for a total of 1 hour (60 minutes).
  - His speed increases with each minute
    - In the 1st minute, he walks at 1 meter per minute,
    - In the 2nd minute, he walks at 2 meters per minute, and so on.
  - Calculate the total distance he walks.
  - Ans: 1830 meters

for Loop

- Scenario 2:
  - John starts walking from his house.
  - He walks for a total of 1 hour (60 minutes).
  - His speed increases with every 2 minute
    - In the 1st minute, he walks at 1 meter per minute,
    - In the 2nd minute, he walks at 1 meters per minute,
    - In the 3rd minute, he walks at 3 meter per minute,
    - In the 4th minute, he walks at 3 meters per minute,
    - and so on.
  - Calculate the total distance he walks.
  - Ans: 1800 meters

Dead loop

- loop over the items of a sequence
- How can we write a dead loop in 'for'?

```
[23] for i in range(4):
         i-=1
         print(i)
```

```
-1
0
1
2
```

Dead loop

```python
ll = [0,1,2,3]
for a in ll:
  ll.append(a+1)
  print(ll)
```

```
[0, 1, 2, 3, 1]
[0, 1, 2, 3, 1, 2]
[0, 1, 2, 3, 1, 2, 3]
[0, 1, 2, 3, 1, 2, 3, 4]
[0, 1, 2, 3, 1, 2, 3, 4, 2]
[0, 1, 2, 3, 1, 2, 3, 4, 2, 3]
[0, 1, 2, 3, 1, 2, 3, 4, 2, 3, 4]
[0, 1, 2, 3, 1, 2, 3, 4, 2, 3, 4, 5]
[0, 1, 2, 3, 1, 2, 3, 4, 2, 3, 4, 5, 3]
[0, 1, 2, 3, 1, 2, 3, 4, 2, 3, 4, 5, 3, 4]
[0, 1, 2, 3, 1, 2, 3, 4, 2, 3, 4, 5, 3, 4, 5]
[0, 1, 2, 3, 1, 2, 3, 4, 2, 3, 4, 5, 3, 4, 5, 6]
[0, 1, 2, 3, 1, 2, 3, 4, 2, 3, 4, 5, 3, 4, 5, 6, 4]
[0, 1, 2, 3, 1, 2, 3, 4, 2, 3, 4, 5, 3, 4, 5, 6, 4, 5]
[0, 1, 2, 3, 1, 2, 3, 4, 2, 3, 4, 5, 3, 4, 5, 6, 4, 5, 6]
[0, 1, 2, 3, 1, 2, 3, 4, 2, 3, 4, 5, 3, 4, 5, 6, 4, 5, 6, 7]
[0, 1, 2, 3, 1, 2, 3, 4, 2, 3, 4, 5, 3, 4, 5, 6, 4, 5, 6, 7, 5]
[0, 1, 2, 3, 1, 2, 3, 4, 2, 3, 4, 5, 3, 4, 5, 6, 4, 5, 6, 7, 5, 6]
[0, 1, 2, 3, 1, 2, 3, 4, 2, 3, 4, 5, 3, 4, 5, 6, 4, 5, 6, 7, 5, 6, 7]
[0, 1, 2, 3, 1, 2, 3, 4, 2, 3, 4, 5, 3, 4, 5, 6, 4, 5, 6, 7, 5, 6, 7, 8]
[0, 1, 2, 3, 1, 2, 3, 4, 2, 3, 4, 5, 3, 4, 5, 6, 4, 5, 6, 7, 5, 6, 7, 8, 6]
[0, 1, 2, 3, 1, 2, 3, 4, 2, 3, 4, 5, 3, 4, 5, 6, 4, 5, 6, 7, 5, 6, 7, 8, 6, 7]
[0, 1, 2, 3, 1, 2, 3, 4, 2, 3, 4, 5, 3, 4, 5, 6, 4, 5, 6, 7, 5, 6, 7, 8, 6, 7, 8]
[0, 1, 2, 3, 1, 2, 3, 4, 2, 3, 4, 5, 3, 4, 5, 6, 4, 5, 6, 7, 5, 6, 7, 8, 6, 7, 8, 9]
[0, 1, 2, 3, 1, 2, 3, 4, 2, 3, 4, 5, 3, 4, 5, 6, 4, 5, 6, 7, 5, 6, 7, 8, 6, 7, 8, 9, 7]
[0, 1, 2, 3, 1, 2, 3, 4, 2, 3, 4, 5, 3, 4, 5, 6, 4, 5, 6, 7, 5, 6, 7, 8, 6, 7, 8, 9, 7, 8]
[0, 1, 2, 3, 1, 2, 3, 4, 2, 3, 4, 5, 3, 4, 5, 6, 4, 5, 6, 7, 5, 6, 7, 8, 6, 7, 8, 9, 7, 8, 9]
```

Break Statement

- Stop the loop when some condition is satisfied
- Can be used in both Python while and for loops

```
for i in range(10):
    print(i)
    if i == 5:
        break
```

```
0
1
2
3
4
5
```

Continue Statement

- Skips the remaining statements in the current loop and starts the next iteration.
- Can be used in both Python while and for loops
  - Example:
  - I don't do exercise on Sunday

```
def PushDown():
    pass
def PushUp():
    pass
pushup_count = 0
day = 0
while pushup_count < 20:
    day += 1
    if day%7==0:
      print(f'day {day}, pushup_count {pushup_count}')
      continue
    PushDown()
    PushUp()
    pushup_count += 1
    print(f'day {day}, pushup_count {pushup_count}')
print("Completed 20 push-ups!")
print(pushup_count)
```

Review

- Loop over the items of a sequence

```
for iterating_var in sequence:
    Statements
```

- Break statement
- Continue statement

Learning a programming language

- Understand the basic vocabulary
- Keywords
  - Words built into language
  - Also called reserved words
- Syntax
  - Rules of language
  - Includes:
    - Spelling
    - Punctuation
    - Grammar
- Each programming language has own unique syntax and structure

Syntax Examples

Syntax:

Python

The syntax of the *if...else* statement is:

```
if expression:
    statement(s)
else:
    statement(s)
```

```
if x > 7 :
    print "x is greater than 7"
else:
    print "x is not greater than 7"
```

If Else

Visual Basic

Syntax

```
If condition Then
        code
    Else
        other code
    End If
```

```
If a > 100 Then
    Console.WriteLine("A is greater than 100")
End If

Console.ReadLine()
```

Syntax Examples

The syntax of an **if...else** statement in C programming language is: **C**

```
if(boolean_expression)
{
   /* statement(s) will execute if the boolean expression is true */
}
else
{
  /* statement(s) will execute if the boolean expression is false */
}
```

```
if( a < 20 )
    {
        printf("a is less than 20\n" );
    }
    else
    {
        printf("a is not less than 20\n" );
    }
```

Syntax Error

- A mistake in a program
- Violates the language rules
- Compiler checks for syntax errors
- Code cannot run unless fixed

```
if True:
print("Hello, World!")

print("Hello, World!"

x = 10
y = "20"
print(x + y)
```

Logic Error

- The syntax is correct
- However, when the code is executed, it produces incorrect results
- Also known as a bug

```
def divide(a, b):
    return a / b

print(divide(10, 2))
print(divide(10, 0))
```

Review

- Some basic concept
  - Syntax Error
  - Logic Error

Input and Output

- Program needs users to provide some data
- Two main different input manners
  - Input from the Keyboard
  - Input from files

Input from the keyboard

```
mark@linux-desktop: /tmp/tutorial
File  Edit  View  Search  Terminal  Help
mark@linux-desktop:~$ mkdir /tmp/tutorial
mark@linux-desktop:~$ cd /tmp/tutorial
mark@linux-desktop:/tmp/tutorial$ mkdir dir1 dir2 dir3
mark@linux-desktop:/tmp/tutorial$ mkdir
mkdir: missing operand
Try 'mkdir --help' for more information.
mark@linux-desktop:/tmp/tutorial$ cd /etc ~/Desktop
bash: cd: too many arguments
mark@linux-desktop:/tmp/tutorial$ ls
dir1  dir2  dir3
mark@linux-desktop:/tmp/tutorial$
```

https://ubuntu.com/tutorials/command-line-for-beginners#4-creating-folders-and-files

input()

- input ()
  - Read one line from standard input
  - Return it as a string
  - We can give a prompt text
    - which will appear before the cursor when you run the code.

```
a = input('tell me are you bored: ')
print(a)
print('tell me are you bored: ')
b = input()
print(b)
```

```
tell me are you bored: no of course
no of course
tell me are you bored:
yes sure
yes sure
```

Type casting

- input ()
  - Read one line from standard input
  - Return it as a string
  - You need to convert the data type if necessary

```
distance_pre = input('how many meters did you walk yesterday? ')
distance_today = input('how many meters did you walk today? ')
total_distance = distance_pre+distance_today
print('oh i know! you walked a total of ')
print(total_distance)
```

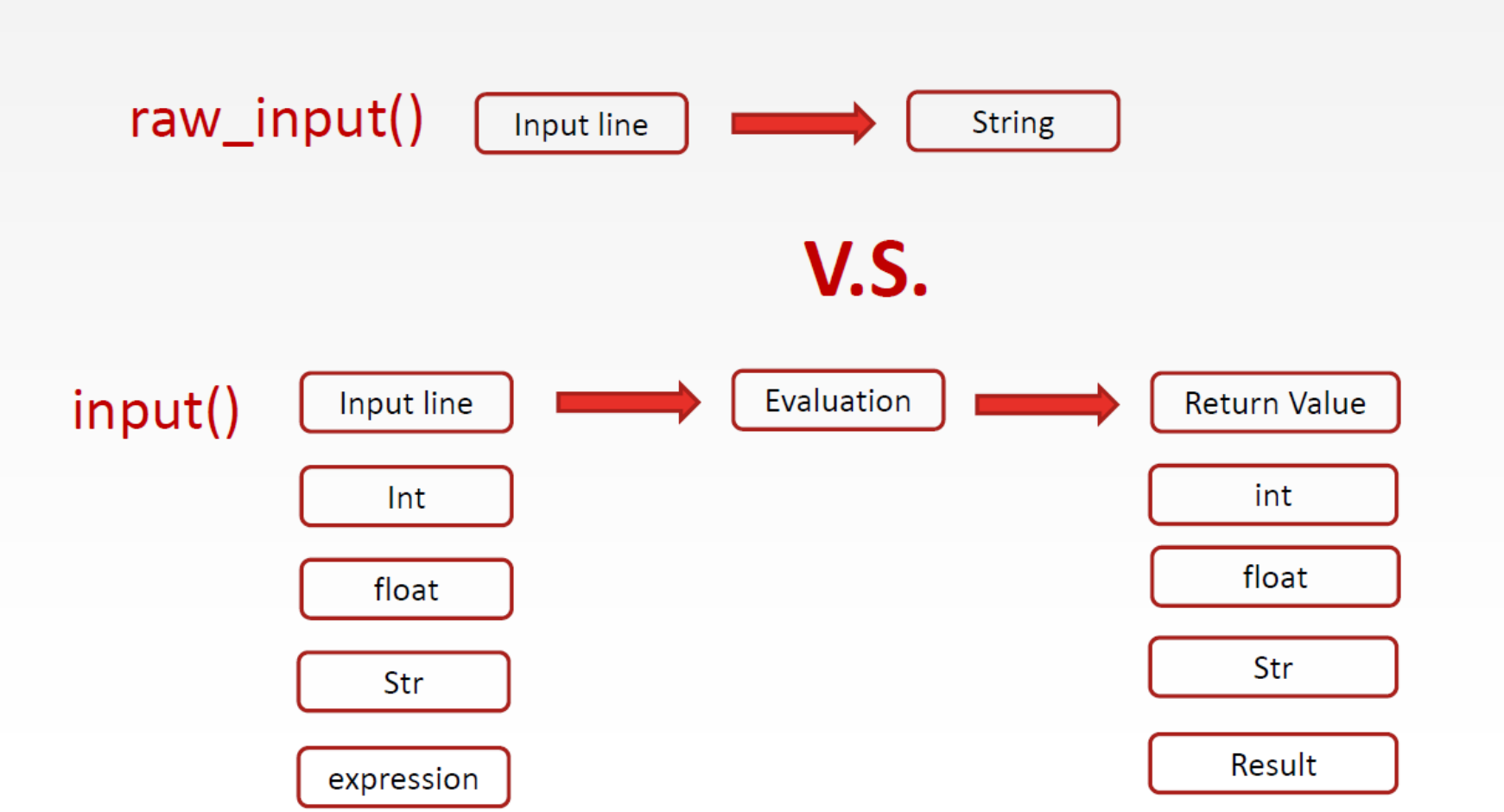
```
how many meters did you walk yesterday? 100
how many meters did you walk today? 200
oh i know! you walked a total of
100200
```

```
distance_pre = input('how many meters did you walk yesterday? ')
distance_today = input('how many meters did you walk today? ')
total_distance = int(distance_pre)+int(distance_today)
print('oh i know! you walked a total of ')
print(total_distance)
```

```
how many meters did you walk yesterday? 100
how many meters did you walk today? 200
oh i know! you walked a total of
300
```

Python2

- raw_input()
- input()



Introduction to sys.argv

- sys.argv is a list in Python
- contains the command-line arguments passed to the script
- sys.argv[0] is the name of the script
- sys.argv[1] to sys.argv[n] are the additional arguments passed
    - >python AverageCalculator.py 3 4 5
    - sys.argv[0] is "AverageCalculator.py"
    - sys.argv[1] is "3"

Output to the screen

- Print one variable's value
  - print ("something or null" + variable_name)
- Print one string variable (e.g. name)
  - print ("this is one string" + name)
- Print one int variable (e.g. age)
  - print (age)
  - print ("Your age is: " + str(age))
- Print one float variable (e.g. salary)
  - print (salary)
  - print ("Your age is: " + str(salary))

Neat and organized print format

```python
tom_salary = 12345.25
jim_salary = 123.5
mike_salary = 12345678.5
jack_salary = 1.23

tom_name = 'Tome Grace Liu'
jim_name = 'Jim'
mike_name = 'Mike Raghu Raghu'
jack_name = 'Jack Li'

print('name    salary')
print(tom_name+'    '+str(tom_salary))
print(jim_name+'    '+str(jim_salary))
print(mike_name+'    '+str(mike_salary))
print(jack_name+'    '+str(jack_salary))
```

```
name    salary
Tome Grace Liu    12345.25
Jim    123.5
Mike Raghu Raghu    12345678.5
Jack Li    1.23
```

Looks better?

```
name     salary
Tome Grace Liu     12345.25
Jim     123.5
Mike Raghu Raghu     12345678.5
Jack Li     1.23
```

```
name                salary
Tome Grace Liu      12345.25
Jim                 123.50
Mike Raghu Raghu    12345678.50
Jack Li             1.23
```

```
                name          salary
     Tome Grace Liu        12345.25
                Jim          123.50
  Mike Raghu Raghu     12345678.50
            Jack Li            1.23
```

Alignment problem
How to align them?

Data Formatting: f-string

- Print variables

```
tom_salary = 12345.25
jim_salary = 123.5
mike_salary = 12345678.5
jack_salary = 1.23

tom_name = 'Tome Grace Liu'
jim_name = 'Jim'
mike_name = 'Mike Raghu Raghu'
jack_name = 'Jack Li'

print('name    salary')
print(tom_name+'    '+str(tom_salary))
print(f'{tom_name}    {tom_salary}')
print(jim_name+'    '+str(jim_salary))
print(f'{jim_name}    {jim_salary}')
```

```
name    salary
Tome Grace Liu    12345.25
Tome Grace Liu    12345.25
Jim    123.5
Jim    123.5
```

f-string

- Print expressions

```
price = 10
quantity = 3
fstring = f'Price: {price}    Quantity : {quantity}    Total : {price*quantity}'
print (fstring)
```

```
Price: 10    Quantity : 3    Total : 30
```

f-string

- Self-debugging

```
price = 10
quantity = 3
for quantity in range(3):
  print('quantity='+str(quantity)+',    Total: price*quantity='+str(price*quantity))
  fstring = f"{quantity=},    Total: {price*quantity=}"
  print (fstring)
```

```
quantity=0,    Total: price*quantity=0
quantity=0,    Total: price*quantity=0
quantity=1,    Total: price*quantity=10
quantity=1,    Total: price*quantity=10
quantity=2,    Total: price*quantity=20
quantity=2,    Total: price*quantity=20
```

f-string

- Alignment
- {variable:>width}
- > right-aligned
- Width: the total number of characters

```python
tom_salary = 12345.25
jim_salary = 123.5
mike_salary = 12345678.5
jack_salary = 1.23

tom_name = 'Tome Grace Liu'
jim_name = 'Jim'
mike_name = 'Mike Raghu Raghu'
jack_name = 'Jack Li'

print('name    salary')
print(f'{tom_name}    {tom_salary}')
print(f'{jim_name}    {jim_salary}')
print('')
print('12345678901234567890    1234567890')
name='name'
salary='salary'
print(f'{name:>20}    {salary:>10}')
print(f'{tom_name:>20}    {tom_salary:>10}')
print(f'{jim_name:>20}    {jim_salary:>10}')
```

```
name    salary
Tome Grace Liu    12345.25
Jim    123.5

12345678901234567890    1234567890
                name        salary
      Tome Grace Liu      12345.25
                 Jim         123.5
```

f-string

- Alignment
- {variable:>width}
- {variable:<width}
- {variable:^width}
- > right-aligned
- < left-aligned
- ^ caret sign, center-aligned
- Width: the total number of characters

```python
tom_salary = 12345.25
jim_salary = 123.5
mike_salary = 12345678.5
jack_salary = 1.23

tom_name = 'Tome Grace Liu'
jim_name = 'Jim'
mike_name = 'Mike Raghu Raghu'
jack_name = 'Jack Li'

print('name    salary')
print(f'{tom_name}    {tom_salary}')
print(f'{jim_name}    {jim_salary}')
print('')
print('12345678901234567890    1234567890')
name='name'
salary='salary'
print(f'{name:>20}    {salary:>10}')
print(f'{tom_name:>20}    {tom_salary:>10}')
print(f'{jim_name:>20}    {jim_salary:>10}')
```

```
name    salary
Tome Grace Liu    12345.25
Jim    123.5

12345678901234567890    1234567890
                name        salary
      Tome Grace Liu      12345.25
                 Jim         123.5
```

```python
tom_salary = 12345.25
jim_salary = 123.5
mike_salary = 12345678.5
jack_salary = 1.23

tom_name = 'Tome Grace Liu'
jim_name = 'Jim'
mike_name = 'Mike Raghu Raghu'
jack_name = 'Jack Li'

print('name    salary')
print(f'{tom_name}    {tom_salary}')
print(f'{jim_name}    {jim_salary}')
print('')
print('12345678901234567890    1234567890')
name='name'
salary='salary'
print(f'{name:<20}    {salary:<10}')
print(f'{tom_name:<20}    {tom_salary:<10}')
print(f'{jim_name:<20}    {jim_salary:<10}')
```

```
name    salary
Tome Grace Liu    12345.25
Jim    123.5

12345678901234567890    1234567890
name                    salary
Tome Grace Liu          12345.25
Jim                     123.5
```

f-string

- Precision handling of floats
- {variable:>[width].[precision]f}
- Width: total number of digits
- Precision: decimal digits
- Do not forget f
- What will happen if we lose f?

```
salary = 543.255
print('your salary is 1234567890, right?')
print(f'your salary is {salary:<10.2f}, right?')
print(f'your salary is {salary:>10.2f}, right?')
print(f'your salary is {salary:>10.4f}, right?')
salary = 3.1415
print(f'your salary is {salary:>10.3f}, right?')
```

```
your salary is 1234567890, right?
your salary is 543.25    , right?
your salary is     543.25, right?
your salary is   543.2550, right?
your salary is      3.142, right?
```

Data Formatting

- Format strings
  - %<width>**s**

```
name = 'Mike'
print('your name is 1234567, right?')
print('your name is %7s, right?'%name)
```

```
your name is 1234567, right?
your name is    Mike, right?
```

- Format integers
  - %<width>**d**

```
age = 80
print('your age is 12345, right?')
print('your age is %5d, right?'%age)
```

```
your age is 12345, right?
your age is    80, right?
```

Data Formatting

- Format a floating point number
  - %<width>.<precision>f
  - Width: total length of the number, including the decimal point
  - Precision: number of decimal places
  - If width < real_length, Print real_length

```
salary = 543.21
print('your salary is 12345678, right?')
print('your salary is %8.3f, right?'%salary)
```

```
your salary is 12345678, right?
your salary is  543.210, right?
```

```
salary = 543.28
print('your salary is 12345678, right?')
print('your salary is %8.1f, right?'%salary)
```

```
your salary is 12345678, right?
your salary is    543.3, right?
```

```
salary = 543.26
print('your salary is 12345678, right?')
print('your salary is %2.2f, right?'%salary)
```

```
your salary is 12345678, right?
your salary is 543.26, right?
```

Further Reading

round(*number, ndigits=None*)

Return *number* rounded to *ndigits* precision after the decimal point. If *ndigits* is omitted or is `None`, it returns the nearest integer to its input.

For the built-in types supporting `round()`, values are rounded to the closest multiple of 10 to the power minus *ndigits*; if two multiples are equally close, rounding is done toward the even choice (so, for example, both `round(0.5)` and `round(-0.5)` are `0`, and `round(1.5)` is `2`). Any integer value is valid for *ndigits* (positive, zero, or negative). The return value is an integer if *ndigits* is omitted or `None`. Otherwise, the return value has the same type as *number*.

For a general Python object `number`, `round` delegates to `number.__round__`.

> **Note:** The behavior of `round()` for floats can be surprising: for example, `round(2.675, 2)` gives `2.67` instead of the expected `2.68`. This is not a bug: it's a result of the fact that most decimal fractions can't be represented exactly as a float. See Floating-Point Arithmetic: Issues and Limitations for more information.

```python
a = 2.675
print(f'{a=}')
b = f'{a:.2f}'
print(f'round a {b}')
a = 2.6751
print(f'{a=}')
b = f'{a:.2f}'
print(f'round a {b}')
print(2.675>2.6749999999999999)
```

```
a=2.675
round a 2.67
a=2.6751
round a 2.68
False
```

How is a decimal represented in a computer?

https://docs.python.org/3/library/functions.html#round

Review

- Input from screen
- Output to screen
- f-string
  - Alignment
  - String width control
  - Decimal precision control

Git

- Git and GitHub are different
- Version Control
  - Forget what has been changed two weeks ago
  - How to combine many people's work
- Go to https://git-scm.com/downloads

Git

1. Go to the command line
   - Windows + R, input *cmd*
2. git config --global user.name "Mona Lisa"
   - If your name has a whitespace, then the double quotes are necessary
3. git config --global user.email "your.email@example.com"
   - Not for logging into any website
   - To record who submit the codes

Git

1. Go to a folder
   - cd d:\
   - cd ~
2. git init project1
   - Will create a folder named project1
3. cd project1
4. Make a txt file: file1.txt
   - Write something and save and close
5. git add file1.txt
6. git commit –m "my first commit"

Git

- Create a public repository on GitHub
- git clone <repository_url>
- https://github.com/ZhengchenZhang/python-lecture2.git
- git branch
- git checkout –b main
- Modify the files
- git add *.py
- git commit –m "my homework"
- git remote add origin <repository_url>
- git push origin main

Secure Shell Protocol (SSH)

1. ssh-keygen -t rsa -b 4096 -C your_email@example.com
2. Type *enter* many times
3. C:\users\<user_name>\.ssh\id_rsa.pub
4. Open it using a text editor, copy the content
5. Go to https://github.com/settings/keys
6. Click *New SSH key*
7. Give any name, and paste the content
8. Get the ssh address of your repository
9. Repeat the comments in the last page