STMC HKOI Training

Lesson 2: Input, variables and data types

Chan Yan Mong

September 29, 2021

Goal today

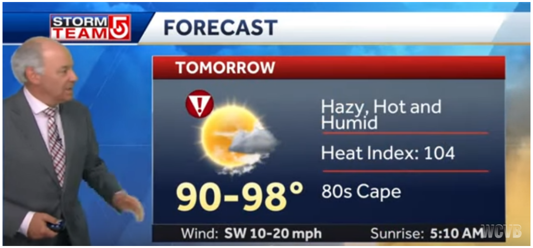
- Reading input with `input`
- Introduce the concept of variable
- Introduce concept of data type: `int`, `float`, `char`, `str`
- Perform basic arithmetics with numerical values to complete a temperature converter

Input, process and output

- Almost all program consist of 3 parts: Recieving inputs, processing inputs, and returning outputs
- For example, in a computer game, the game continuously recieve inputs like mouse clicks, keyboard typing, process them to perform actions on the character, and finally displaying the updated screen to you
- Today we will like to illustrate concepts surrounding these things using an example

Example: Temperature conversion

- We usually use SI units in our data life
- But some places (like the U.S.) uses imperial units



Source: https://youtu.be/Ke9niDoM3f8

Example: Temperature conversion

- In imperial units, temperature are represented using Fahrenheit (^{o}F)
- In this system, 0^oF is defined to be the freezing point of a solution of brine and 96^oF is defined to be average human body temperature
- To convert Fahrenheit to Celsius, one will need to use the following (inconvenient) formula:

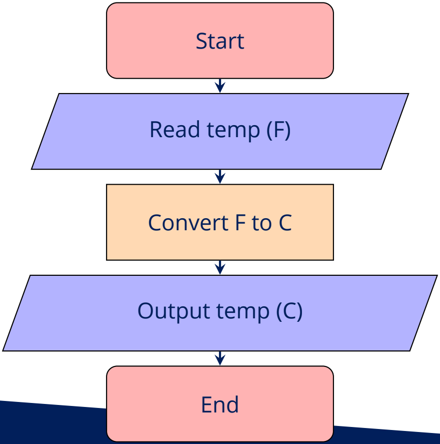
$$T_C = \frac{5}{9}\left(T_F - 32.0\right)$$

- For example, in the picture above, the temperature $90^o\text{F} = \frac{5}{9}(90-32) \approx 32^o\text{C}$
- As you can see, its' pretty cumbersome. So let's write a program to do it for us

Designing a program

- Let’s try to design the program
- First of all, let’s identify the input, process and output:
  - Input: Degree in Fahrenheit
  - Process: Convert Fahrenheit to Celsius using formula
  - Output: Print Degree in Celsius
- This can be summarized using a **flowchart**, which describe the flow of the program

Flow chart



Let’s implement

- Now we know how the program flows, let’s put our ideas in code!
  - Input: We don’t know how to do
  - Process: We don’t know how to do
  - Output: We already know how to do with `print`
- By the way, the action of converting ideas to code is sometimes called **implementation**

Reading inputs

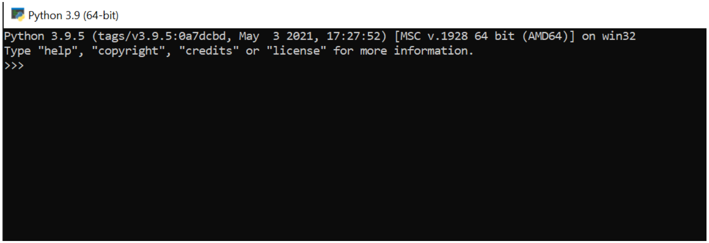
- Inputs in python are read using `raw_input` and `input` function
- We shall first try `input`
- According to the documentation,the syntax of `input` is:

```
input(prompt)
```

where `prompt` is a string representing a default message before the input.

Reading input

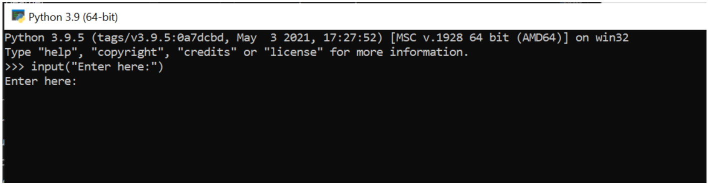
- What does that even mean?
- Let’s experiment by opening up the python interactive console



```
Python 3.9.5 (tags/v3.9.5:0a7dcbd, May  3 2021, 17:27:52) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Reading input

- Type in `input("Enter here:")` and enter
- The following should show:

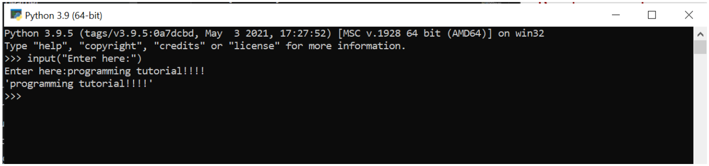


Python 3.9 (64-bit)

```
Python 3.9.5 (tags/v3.9.5:0a7dcbd, May  3 2021, 17:27:52) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> input("Enter here:")
Enter here:
```

Reading input

- Type something and press enter
- You should see your input displaying:



```
Python 3.9.5 (tags/v3.9.5:0a7dcbd, May  3 2021, 17:27:52) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> input("Enter here:")
Enter here:programming tutorial!!!!
'programming tutorial!!!!'
>>>
```

Reading input

- As you can probably see, `input` ask you a question which you can answer
- You answer by typing some input and press enter to submit
- Furthermore, the text inside `input` is displayed when it prompts for answer
- So you can imagine in our final program the code would have something like this:

```
        input("Enter temperature in (F): ")
```

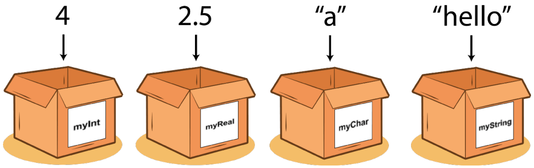
- But here is a problem: How can we save and manipulate these input?

Why do we need to store?

- But you may ask, why do we need to save results anyway?
- The answer is actually quite obvious: *If you don't remember anything, you can't process anything*
- *Example: Guy who forgot everything he heard*
- *Imagine asking a forgetful guy what is the difference in age between you and your friends. One by one, you and your friend told the guy your ages. Being forgetful, he never remembers anything he heard. You can see such a guy will never be able to answer your question, because he would have forgotten your age by the time he listen to your friend's!*

Variables - Your handy storage box

- To store data, a computer uses something called a **variable**
- A variable is a ”storage box” with a name
- It stores data temporarily so that the value inside can be retrieved for further processing



Source: https://stevenpcurtis.medium.com/what-is-a-variable-3447ac1331b9

Three parts of a variable

- A variable always consist of 3 things:
  1. **Name** for us to refer later in the program
  2. **Value** that contain what is stored
  3. **Data type** that indicate what *kinds* of value is stored
- Name and value is trivial, let’s look at data types

What is data types?

- Values in computer are always stored as 0s and 1s
- The same sequence of bits can represent different things according to the way we decode it!
- Different data type care stored and manipulate differently by computer!
- So important to let the computer knows what **data types** they are dealing with

Data types: Integer

- **Integers** ($\mathbb{Z}$) are *numbers without decimal points and fraction component*
- Examples of integers $\{\cdots, -4, -3, -2, -1, 0, +1, +2, +3, +4, \cdots\}$
- Examples of integral data: *age*, *no. of apples*, *no. of people*

Data types: Float

- **Floating point numbers** are *numbers with decimal points*
- Examples of floating point numbers: 2.5, 0.0, -3.1, 100.173
- Examples of float data: *temperature*, *volume*, *mass*, *height*

Data types: Character

- **Character** is *a single character*
- Usually quoted by **single quotes (’’)**
- Examples of characters numbers: `'a','F','<space>','@','0','+'`
- Examples of character data: *grades*

Data types: String

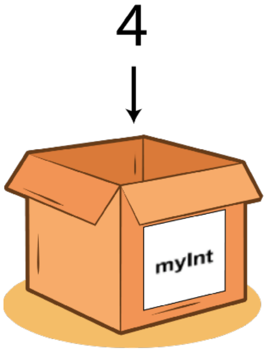
- **String** is *a sequence of one or more character*
- Usually quoted by **double quotes ("")**
- Examples of string: `"hello","ymchan@gmail.com","1+2=3","kim_979"`
- Examples of string data: *name*, *email*, *username*, *password*

Data types: Boolean

- **Boolean** is a variable either `true` or `false`
- Usually denote a state, or some flag
- Examples of boolean data: *is_married*, *is_empty*, *is_opened*, *is_running*

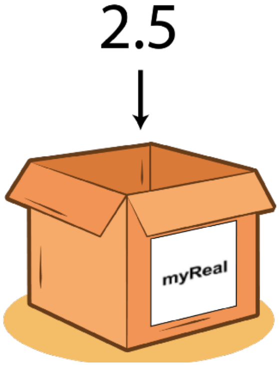
Case study I

- Variable name is **myInt**
- The value stored is **4**
- The data are numbers *without decimal points*, so the data type is **integer**



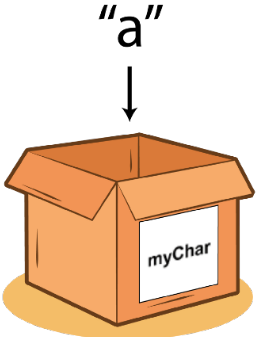
Case study II

- Variable name is **myReal**
- The value stored is **2.5**
- The data are numbers *with decimal points*, so the data type is **floating point number**



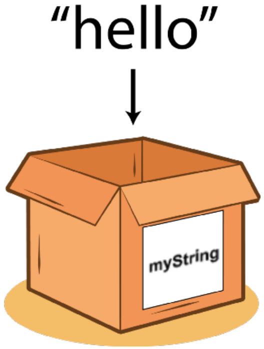
Case study III

- Variable name is **myChar**
- The value stored is **"a"**
- The data is a *single character*; The data type is **character**



Case study IV

- Variable name is **myString**
- The value stored is **"hello"**
- The data is a *sequence of characters*; The data type is **string**



Declaring variables

- Let’s see how all these turn into code:
- To **declare** a variable, do the following

```
    <variable name> = <value>
```

- For example:

```
      myName = "John" # String variable myName storing "John"

      age = 19 # Integer variable age storing 19

      temp = 23.3 # Float variable called temp storing 23.3

      isOk = False # Boolean variable called isOK storing False
```

Accessing values

- After declaration, we can access the values stored by the variable name:

```python
name = "John"
# This will output "Your name is: John"
print("Your name is:", name)

temp = 22.1
temp + 1 # 23.1
temp - 10 # 13.1
# Output: "Temp minus 10: 13.1"
print("Temp minus 10: ", temp - 10)
```

Checking data type

- We can also look at the data type of things using `type()` function

```
>>> type(1)
<class 'int'>

>>> type(1.2)
<class 'float'>

>>> type('12')
<class 'str'>

>>> type(True)
<class 'bool'>
```

Checking data type

- Similarly for variables

```
>>> myName = "John"
>>> type(myName)
<class 'str'>

>>> temp = 30.2
>>> type(temp)
<class 'float'>
```

Checking data type

- We can also use the `is` keyword to check if an object belongs to a type

```
>>> age = 20
>>> type(age)
<class 'int'>
>>> type(age) is int
True

>>> myName  = "rs132"
>>> type(myName) is str
True

>>> isDone = False
>>> type(isDone) is bool
True
>>> type(isDone) is int
False
```

Doing operations with variables

- We can also manipulate the values stored in the variable using various operators. For example, if the varaibles are `int` or `float`:

```
>>> 1.0 + 2.0      # Addition
3.0
>>> 3 - 10         # Subtraction
-7
>>> 3 * 5          # Multiplication
15
>>> 4/3            # Division
1.3333333333333333
>>> 4**5           # Exponent (4**5 = 4*4*4*4*4)
1024
```

Doing operations with variables

- Similarly for string, we have the + and ∗ operations defined
- Note that some operations will results in error

```
>>> "hello" + "bye"        # Concatenate two string
'hellobye'
>>> "hello"*3              # Concatenate string 3 times
'hellohellohello'

>>> "hello" * "bye"        # Make no sense
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cant multiply sequence by non-int of type 'str'

>>> "hello" - "bye"        # Make no sense 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for -: 'str' and 'str'
```

Saving results

- Finally, we can **assign** our computational results back to string
- This is done using the = operator

```
>>> i = 5
>>> i + 9         # Won't change i
14
>>> i
5
>>> i = i + 9     # i = xxx Set i to be xxx
>>> i
14
```

Combining everything

- Combining everything, this give us the following code:

```
tempF = input("Enter temperature in (F): ")
tempC = 5.0*(tempF - 32.0)/9.0
print("Temperature in (C): ",tempC)
```

- Let's see how it works out... Ops! An error!

```
>>> python3 temp.py
Enter temperature in (F): 80
Traceback (most recent call last):
  File "temp.py", line 2, in <module>
    tempC = 5.0*(tempF-32.0)/9.0
TypeError: unsupported operand type(s) for -: 'str' and 'float'
```

What is the error?

- What is the error? Let’s look at the **error message** more closely:

```
    File "temp.py", line 2, in <module>
    tempC = 5.0*(tempF-32.0)/9.0
TypeError: unsupported operand type(s) for -: 'str' and 'float'
```

- According to the message, we are using the subtraction operand (−) to subtract `str` and `float`!
- But why? Isn’t our input 80 a float?

What is the error?

- Is it really so? In debugging, it's always good to check our assumptions because sometimes it can be wrong!
- Let's check by printing the type of `tempF`

```python
tempF = input("Enter temperature in (F): ")
print('Type of tempF: ',type(tempF))
tempC = 5.0*(tempF - 32.0)/9.0
print("Temperature in (C): ",tempC)
```

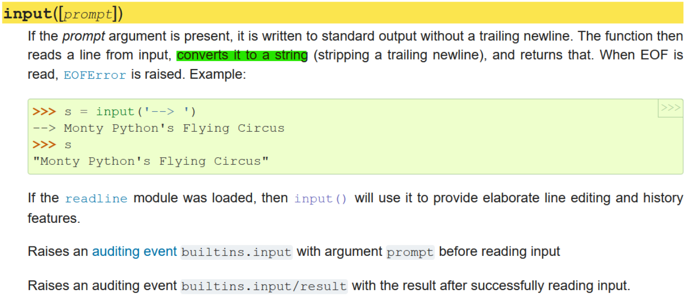
What is the error?

- Let’s see what it returns now:

```
>>> python3 temp.py
Enter temperature in (F): 80
Type of tempF:  <class 'str'>    <---- Ops! tempF is a string!!!
Traceback (most recent call last):
  File "temp.py", line 3, in <module>
    tempC = 5.0*(tempF-32.0)/9.0
TypeError: unsupported operand type(s) for -: 'str' and 'float'
```

What is the error?

- Turns out, the problem is the way `input` handles input. According to the documentation:



input([*prompt*])

If the *prompt* argument is present, it is written to standard output without a trailing newline. The function then reads a line from input, converts it to a string (stripping a trailing newline), and returns that. When EOF is read, `EOFError` is raised. Example:

```
>>> s = input('--> ')
--> Monty Python's Flying Circus
>>> s
"Monty Python's Flying Circus"
```

If the `readline` module was loaded, then `input()` will use it to provide elaborate line editing and history features.

Raises an auditing event `builtins.input` with argument `prompt` before reading input

Raises an auditing event `builtins.input/result` with the result after successfully reading input.

What is the error?

- So basically, `input` will convert everything we input, regardless of it’s original form, into string!
- To battle this, we need to *manually* turn `str` back to `float`
- This is done by wrapping `float()` in front of `input`, i.e. from:

```
input("Enter temperature in (F): ")
```

to

```
float(input("Enter temperature in (F): "))
```

Final code

- The final code is then:

```
tempF = float(input("Enter temperature in (F): "))
print('Type of tempF: ',type(tempF))
tempC = 5.0*(tempF - 32.0)/9.0
print("Temperature in (C): ",tempC)
```

- and finally we can run!

```
>>> python3 temp.py
Enter temperature in (F): 80
Type of tempF:  <class 'float'>
Temperature in (C):  26.666666666666668
```

Some comment on type casting

- The trick we used to solve the problem is actually called **type casting**
- Type casting converts data type from one kinds to another
- In general, to *cast* a value / variable to `<data-type>`, do:

```
<data-type>(<value/variable>)
```

- For example:

```
float("1.2")     # Converts string "1.2" to float
int("12")        # Converts string "12" to int
str(1.2)         # Converts float 1.2 to string
```

Radius from circumference

- Write a program that takes in the circumference of a circle and output the radius of the circle
- **Solution**:

```
C  = float(input("Enter circumference")) # Circumference
PI = 3.1415926535 # Pi
r = C/(2*PI)      # Circumference/2 Pi
print("Radius is: ",r)
```

Swapping numbers

- Complete the following code that swap the values in `i` and `j`

```python
i = 4
j = 6
print("i: ",i, " j: ",j)       # i: 4 j: 6
# Do something here to swap i,j. Can define new variable
print("i: ",i, " j: ",j)       # i: 6 j: 4
```

- **Solution:**

```python
temp = i
i = j
j = temp
```

Length converter

Problem

We will write something similar to the Fahrenheit to Celsius converter. Instead of converting temperature, we will write code that convert length from miles to kilometer. Write a program that read in length in miles, then output length in kilometer. (Remark: 1 miles $\approx$ 1.60934 km)

Expected output

```
Please enter length in miles: 2.7
Length in kilometer: 4.345218
```

Average Calculator

Problem

Write a program that takes in 5 numbers $x_1, x_2, \cdots, x_5$ and calculate the average of the numbers $\bar{x} = (x_1 + x_2 + x_3 + x_4 + x_5)/5$

Expected output

```
2.5
2.1
4.5
1.2
4.5
Average of these numbers is 2.96
```

Challenging: Number of hand shakes

Problem

n business people meet for lunch and shake hands with each other. How many handshakes are there? (Obviously $n \in \mathbb{Z}^+$)
(Hint: Consider the case when $n = 1, 2, 3, 4, 5$, can you see a pattern?)

Challenging: Number of hand shakes (Answer)

Answer

Let n be the number of people and S_n be the number of handshakes.

n	1	2	3	4	5	6	7
S_n	0	1	3	6	10	15	21

Observe that $S_n = S_{n-1} + (n-1)$ for $n > 1$

Challenging: Number of handshakes (Answer)

Thefore:

$$\begin{aligned}
S_n &= S_{n-1} + (n-1) \\
&= (S_{n-2} + (n-2)) + (n-1) \\
&= (S_{n-3} + (n-3)) + (n-2) + (n-1) \\
&= \cdots \\
&= (S_2 + 2) + 3 + \cdots + (n-2) + (n-1) \\
&= (S_1 + 1) + 2 + 3 + \cdots + (n-2) + (n-1) \\
&= \mathbf{0} + \mathbf{1} + \mathbf{2} + \mathbf{3} + \cdots + (\mathbf{n} - \mathbf{2}) + (\mathbf{n} - \mathbf{1})
\end{aligned}$$

Challenging: Number of handshakes (Answer)

$$S_n = 0 + 1 + 2 + 3 + \cdots + (n-2) + (n-1)$$

- In fact, it can be shown that $S_n = n(n-1)/2$
- This is because each people shakes with $n-1$ people
- Since there are n peoples, the number of shakes should be $n(n-1)$
- But A shakes B and B shakes A are count the same, so we double counted
- Therefore $S_n = n(n-1)/2$

Challenging: Number of handshakes (Answer)

Implementation in code:

```
n = input('Enter number of people: ')
n = int(n)
Sn = n*(n-1)//2 # // for integer division
print('Total number of handshakes: ', Sn)
```