HKUST Future-Ready Scholars

Introduction to Game Programming using Python

Additional Content for Workshop 1

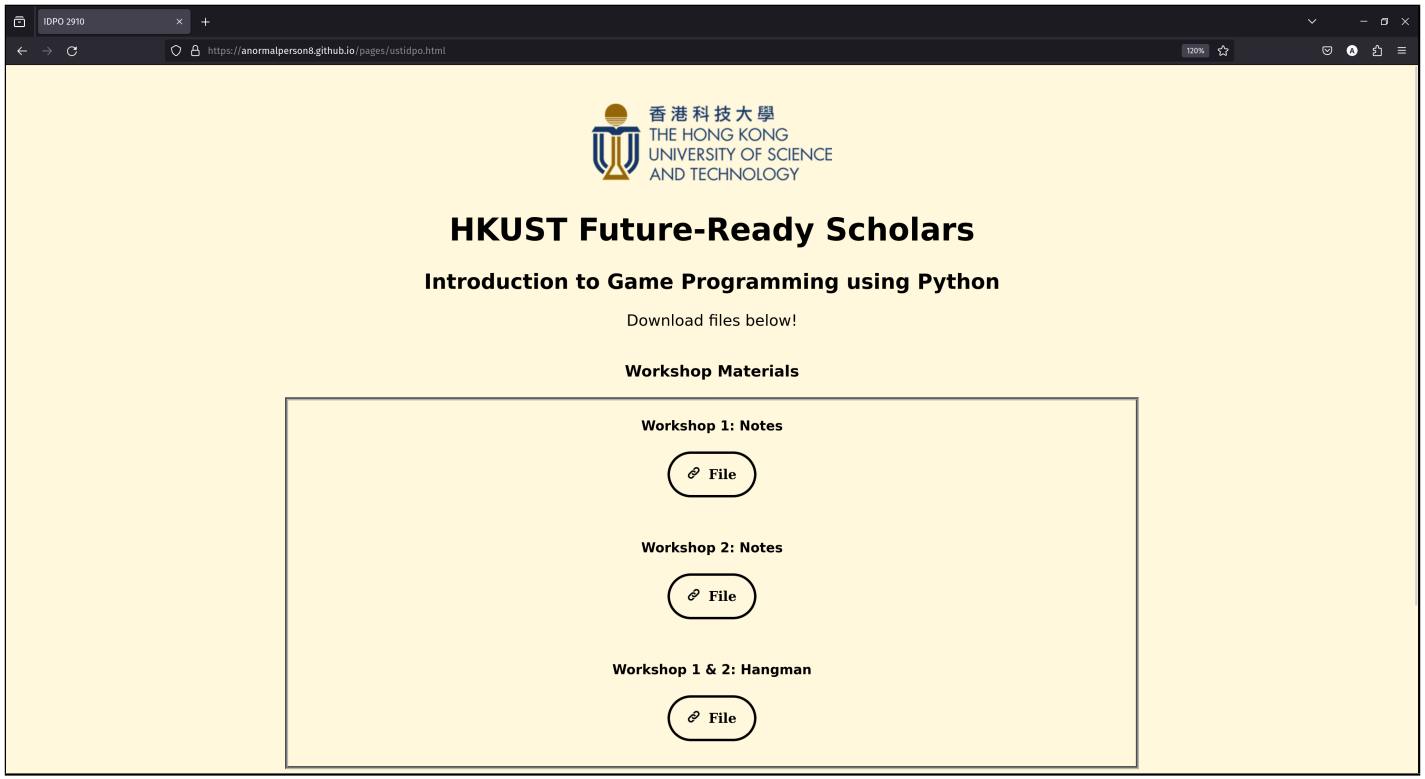
20 April 2024

Google Colab

We will use Google Colab for the workshops.
https://colab.research.google.com/
You must have a Gmail account for it, create one if you do not.

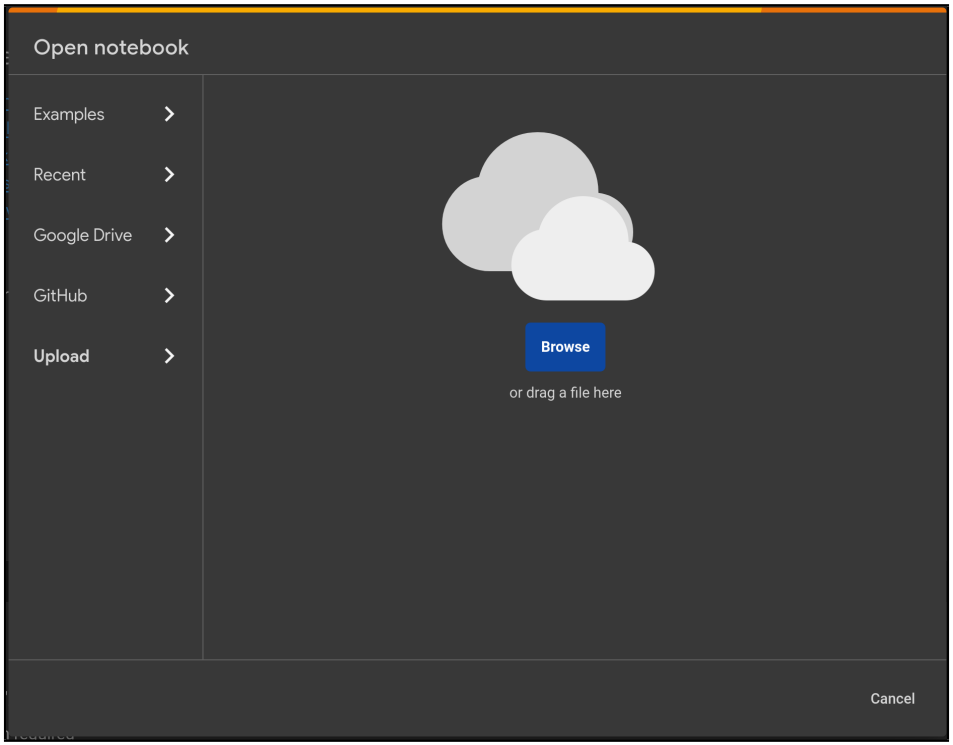
Files

All materials are at:
https://bit.ly/ustidpo



Jupyter Notebook

You can upload your Jupyter Notebook file with **Files** → **Open Notebook**.



Using Jupyter Notebook

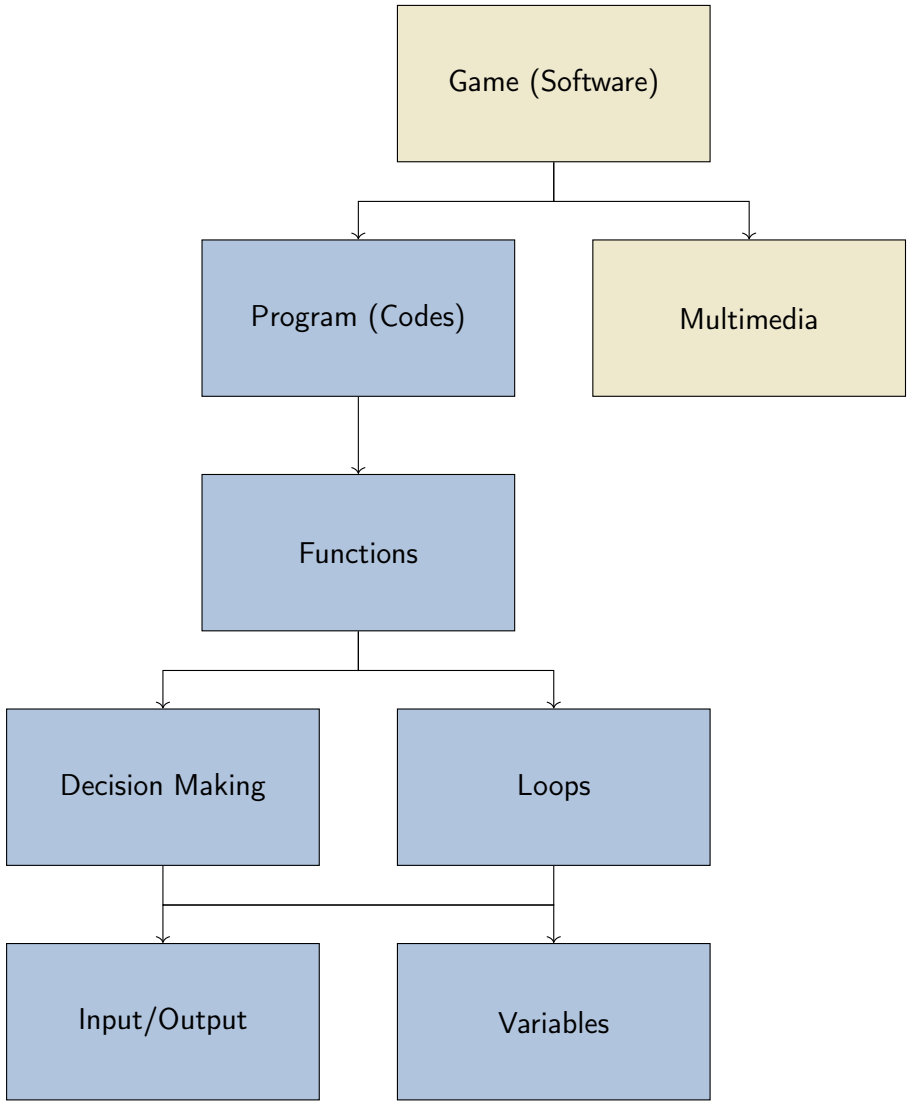
You can type your code in these blocks. We call these blocks code cells.

You can run a code cell with the button on the left.

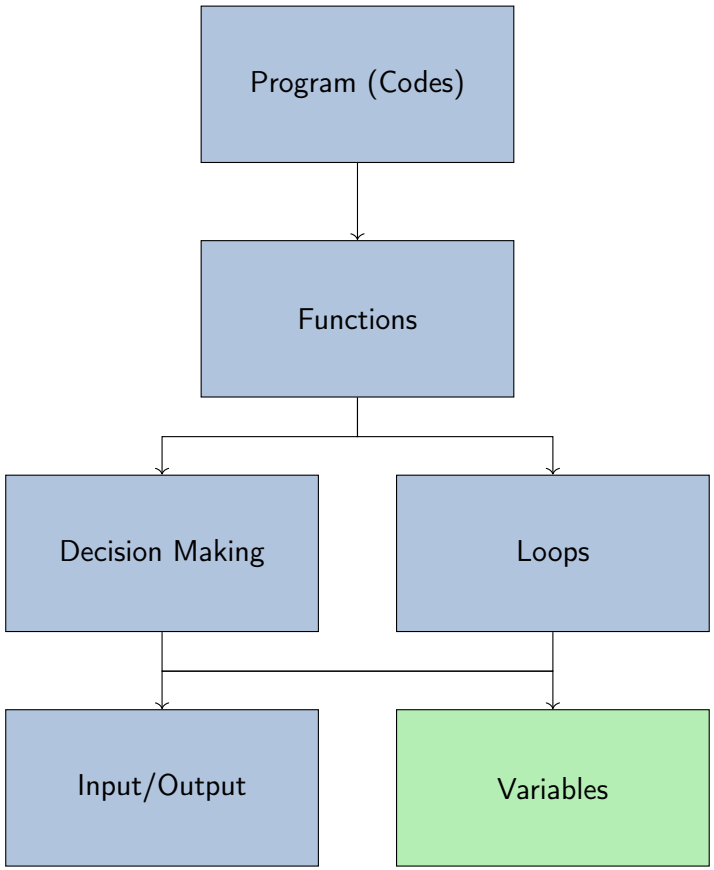
This set of notes is purely **supplementary** to workshop 1, for those who want to learn more and get a more complete picture of the Python programming language.

This is the logo of Python.

World of Game Coding



Contents


Program (Codes)
Functions
Decision Making
Loops
Input/Output
Variables

Variables

Imagine you borrow a box from the computer.

Give it a name and a value, you can now recall this value with the name!

Variables

The code usually goes:
`variable_name = data`
This means whatever `data` is, it is now stored in a variable with name `variable_name`.

In these workshops we will only deal with integer numbers and strings.
Integers: numbers WITHOUT decimal points.
String: Words/Letters surrounded by (single OR double) quotation marks.

```
box_a = 5              # Integer
box_b = -100           # Integer
box_c = "Some text" # String
box_d = 'Some more' # String
```

Variable Types

Some basic variable types:

```
a = 5        # This is an integer (int) stored in a
b = True     # This is a boolean (bool) stored in b
c = 3.2      # This is a float (float)  stored in c
d = "abc"    # This is a string (str)   stored in d
e = 'abc'    # This is also a string    stored in e
```

Variables - Integers

What are integers?
Integers are just like what you've learnt in Maths, numbers without decimal points. Are the following valid?

```
a = 5         # Valid
b = 12        # Valid
c = 1000000 # Valid
d = -1984     # Valid
e = 32.5      # This is NOT an integer (but a float)
f = 5.0       # This is NOT an integer (but a float)
g = '5'       # This would become a string instead
```

Variables - Integer Arithmetic Operations

You can do normal operations on integers:

```
a = 1 + 2    # a stores the integer 3
b = 80 - 52  # b stores the integer 28
c = 69 * -2  # c stores the integer -138
d = 6 / 4    # d stores the float 1.5
e = 18 / 2   # e stores the float 9.0
```

Division in Python

Whether a number can be precisely divided or not, division does NOT return an integer, but a float instead. You can apply type conversion to change it, which is introduced later.

Variables - Integer Arithmetic Operations

Some more examples on operations, with variables:

```
a = 100
b = 12
c = a + b    # c stores the integer 112
d = b - a    # d stores the integer -88
e = a * -b   # e stores the integer -1200
f = a / b    # f stores the float 8.333333333333334
```

Variables - Integer Arithmetic Operations

Then how do we get an integer output?

```
a = 100
b = 12
c = a // b  # c stores the integer 8
            # // operator takes the closest and smaller
            # integer from the division operation
d = a % b   # d stores the integer 4
            # % operator takes the remainder of a
            # division operation
```

Variables - Integer Arithmetic Operations

Also, the power (exponent) operation:

```
a = 2
b = 5
c = a ** b  # c stores the integer 32
            # ** operator means power
```

Variables - Floats

What are floats?
Floats are numbers with decimal points.
Arithmetic operators we learnt can be applied as well.

```
a = 0.2      # a stores the float 0.2
b = 3.0      # b stores the float 3.0
c = a + b    # c stores the float 3.2
d = b / a    # d stores the float 15.0
e = a ** b   # e stores the float 0.008000000000000002
```

Inaccuracies

Inaccuracies happen with decimals in Python. Be careful when dealing with floats.

Variables - Floats

What happens when you combine floats and integers?

```
a = 0.2      # a stores the float 0.2
b = 3        # b stores the integer 3
c = a + b    # c stores the float 3.2
d = b / a    # d stores the float 15.0
e = a ** b   # e stores the float 0.008000000000000002
```

Arithmetic operations between `int` and `float`

Arithmetic operations between integers and floats converts the integer into a float first before operating.

Variables - Boolean values

What are boolean values?

There are only 2 boolean values in existence: `True` and `False`.

```
a = True
b = False
```

Variables - Strings

What are strings?

```
a = "word"  # a stores the string "word"
b = 'word2' # b stores the string "word2"
c = '5.20'  # c stores the string "5.20"
d = 'abc"   # error
```

Quotes

In Python you must use corresponding quotation marks for strings.

Variables - Strings

Example:

```
a = "haha"
b = "hehe"
c = a + b     # c stores the string "hahahehe"
```

Concatenation of strings

You can concatenate (add) strings together with the addition symbol.

Variables - Strings

How do I put the symbols ' and " into a string?
For ":

```
a = "word\"" # a stores the string "word""
b = 'word"'  # b stores the same string as a
```

Same goes for single quotes ':

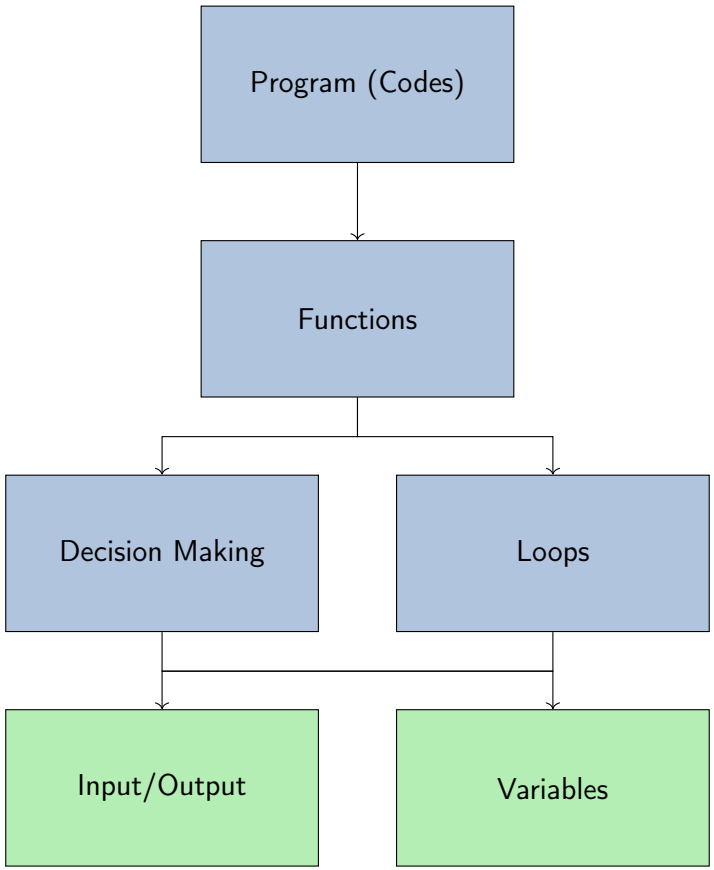
```
a = 'word\'' # a stores the string "word'"
b = "word'"  # b stores the same string as a
```

Variables - Strings

There are additional symbols in strings.

```
a = "word\n" # \n represents the newline character
b = "word\t" # \t represents the tab character
```

Contents


Program (Codes)
Functions
Decision Making
Loops
Input/Output
Variables

The print() function

```
print("This is the print function.")
```

The print() function

print() is a function that lets you print something, also known as text output. What this means is whenever you want to put something on screen it has to be *print*ed out.

```
print("Word") # This prints the word "Word".
```

Examples:

```
>>> print("Hello World")
Hello World
>>> print("Haha hehe")
Haha hehe
>>> print(5)
5
```

Printing multiple things

You can use a comma (,) to separate different things with a space.

```
>>> print("Alpha", "Beta", "Gamma")
Alpha Beta Gamma
>>> print("Haha", "hehe")
Haha hehe
>>> print(19, 91)
19 91
```

Variables in output using the `print()` function

How do we print variables?

```
a = 5
print(a)          # 5
b = "haha"
print(b)          # haha
print(a + 2)      # 7
```

Calculation

We can calculate expressions inside the `print()` function.

Variables in output using the `print()` function

How do we print variables?

```
a = 5
print(a)          # 5
b = "haha"
print(a, b)       # 5 haha
print(b, b)       # haha haha
```

The comma

`,` in `print()` works the same with variables.

Variables in output using the `print()` function

How do we print variables?

```
a = 5
print(a)          # 5
b = "haha"
print(b)          # haha
print(a + "5") # error
print(b + 2)    # error
print(a + b)    # error
```

Addition

You cannot use addition to print things of incompatible types.
`int` and `float` types are not incompatible because all `int` are converted to `float` if needed during operation, as mentioned before.

Variables in output using the `print()` function

How do we print variables?

```
a = 5
b = 32
c = 32.0
print(a * b)     # 160
print(a * c)     # 160.0
```

Takeaway

`print()` function evaluates the expression inside the brackets first before actually printing.

More on `print()` function

In Python, the `print()` function automatically adds a new line after execution. We, however, can stop that.
The `end=` tag allows us to define the character added when `print()` is executed.

```
print(5, end="")
print(4)
print("a", end="abc")
print("d", end=" ")
print("e")
# What is the output?
# Output: 54
#         aabcd e
```

End of line

Remember to include a new line `\n` in the last line of a printed string. Else it may mess up the future outputs from other lines of the code or the computer terminal.

More on `print()` function

We mentioned that whenever `,` is used in `print()`, the items would be separated by a space.
This can actually be changed using the `sep=` tag.

```
>>> print("100", 100, end="\n3\n")
>>> 100 100
    3
>>> print("100", 100, sep="a", end="\n3\n")
>>> 100a100
    3
```

More on `print()` function

Another example:

```
>>> a = 5
>>> b = 10
>>> print(a, b, a + b, end="20\n")
>>> 5 10 1520
>>> print(a, b, a + b, sep="", end="20\n")
>>> 5101520
>>> print(a, b, a + b, end="20\n", sep="")
>>> 5101520
```

Command Parameters

As long as you mark `sep` and `end` clearly **and** after the things you want to print, the ordering doesn't matter!

input() function

We know how to output (print), what about input?

```
input("This is the input function.")
```

input() function

input() is a function that outputs a prompt and lets the user enter something.

```
>>> input("Enter a number: ")
Enter a number: 5
```

Simply inputting doesn't do anything, but we can print it.

```
>>> print(input("Enter a number: "))
Enter a number: 100
100
```

input() function

Another example:

```
>>> input("Enter something: ")
Enter something: I am in HKUST
```

Simply inputting doesn't do anything, but we can print it.

```
>>> print(input("Enter a number: "))
Enter something: I am in HKUST
I am in HKUST
```

Converting the type of an input

How do we convert the data type of variables?

```
>>> number = input("Enter your number: ")
Enter your number: 50
>>> print(number)
50
>>> print(number + 1000) # Error occurs. Why?
```

Explanation

`number` is a string type while 1000 is an integer.

Converting the type of an input

How do we convert the data type of variables?

```
>>> number = input("Enter your number: ")
Enter your number: 50
>>> print(number)
50
>>> print(int(number) + 1000) # 1050
```

Type conversion

`input()` returns the input as string. We need to convert the input to the suitable type when needed.
We use `int()` to convert something into an integer.
This will be useful.

Converting between types

You can convert between types with their type names in Python.

Data Type	Command
Integer	`int()`
Float	`float()`
String	`str()`
Boolean	`bool()`

int()

int() tries to convert a variable into an integer.

```
a = 10          # int
print(int(a)) # 10
                # Nothing occurs

b = 3.7         # float
print(int(b)) # 3
                # Discards values to the right of
                # the decimal point

c = True        # boolean
print(int(c)) # 1

d = False       # boolean
print(int(d)) # 0
                # For boolean: 0 if False, True otherwise
```

int()

```
i = "123abc"  # string
print(int(i)) # Error

j = "123"     # string with ONLY numbers
print(int(j)) # 123
              # Only integers in strings would be
              # successfully converted

k = "123.123" # string with ONLY numbers, but with
              # a number that represents a float
print(int(k)) # Error
```

float()

The concepts of `int()` and `float()` are quite similar.

```
a = 10              # int
print(float(a)) # 10.0
                    # From int -> float

b = 3.7             # float
print(float(b)) # 3.7
                    # Nothing happens

c = True            # boolean
print(float(c)) # 1.0

d = False           # boolean
print(float(d)) # 0.0
```

float()

```
i = "123abc"     # string
print(float(i)) # Error

j = "123"        # string with ONLY numbers
print(float(j)) # 123.0

k = "123.123"    # string with ONLY numbers, but with
                 # a number that represents a float
print(float(k)) # 123.123
```

str()

All of the 3 data types below can be transformed into strings.

```
a = 10          # int
print(str(a)) # 10

b = 3.7         # float
print(str(b)) # 3.7

c = True        # boolean
print(str(c)) # True

d = False       # boolean
print(str(d)) # False
```

str()

```
e = "abcdef"    # string
print(str(e))   # abcdef
                # Nothing happens
```

bool()

```
a = 0              # int
print(bool(a)) # False
                   # 0 means False

b = 3.7            # float
print(bool(b)) # True
```

True and False values

Any integers or floats, if they are not zero, then `bool()` returns `True`, `False` otherwise.

```
c = True           # boolean
print(bool(c)) # True

d = False          # boolean
print(bool(d)) # False
                   # Nothing happens for the 2 above
```

bool()

`bool()`, when applied to a string, checks whether it has content:

```
e = "abcdefg"
print(bool(e)) # True
f = "False"
print(bool(f)) # True
g = " tRuE "
print(bool(g)) # True
h = "0"
print(bool(h)) # True
i = ""
print(bool(i)) # False
```

Strings

If the string has a length > 0, then `bool()` returns `True`, `False` otherwise.

Example of input and type conversion

```
age = int(input("How old are you? "))
print("You are", age, "years old.")
```

Running the program:

```
How old are you? 69
You are 69 years old.
```

Invalid input

If the input does not contain *only* an integer, then the program would throw an error.

Example of input and type conversion

```
age = int(input("How old are you? "))
print("You are", age, "years old.")
```

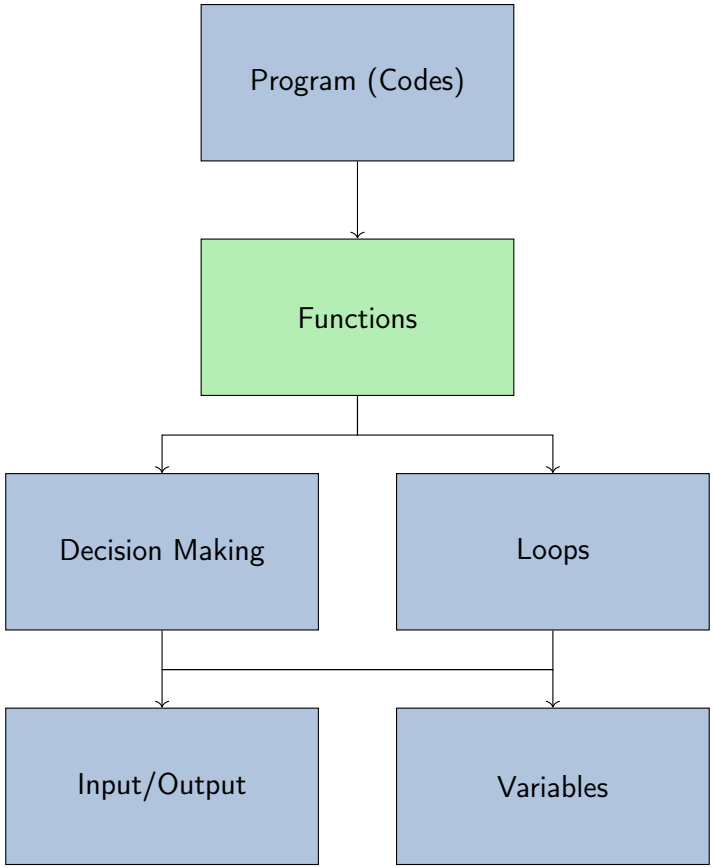
Running the program with an invalid input:

```
How old are you? 69.420
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '69.420'
```

Invalid input

This also applies to data types like boolean values and strings.

Contents



This is a *very* brief peek into functions.

Generating a random integer using `random` library

In Python, we can import libraries to help us with tasks. One of them is generating random numbers. The library/package `random` allows us to get a random number.
The `randint` function provided allows us to generate a random integer given a range.

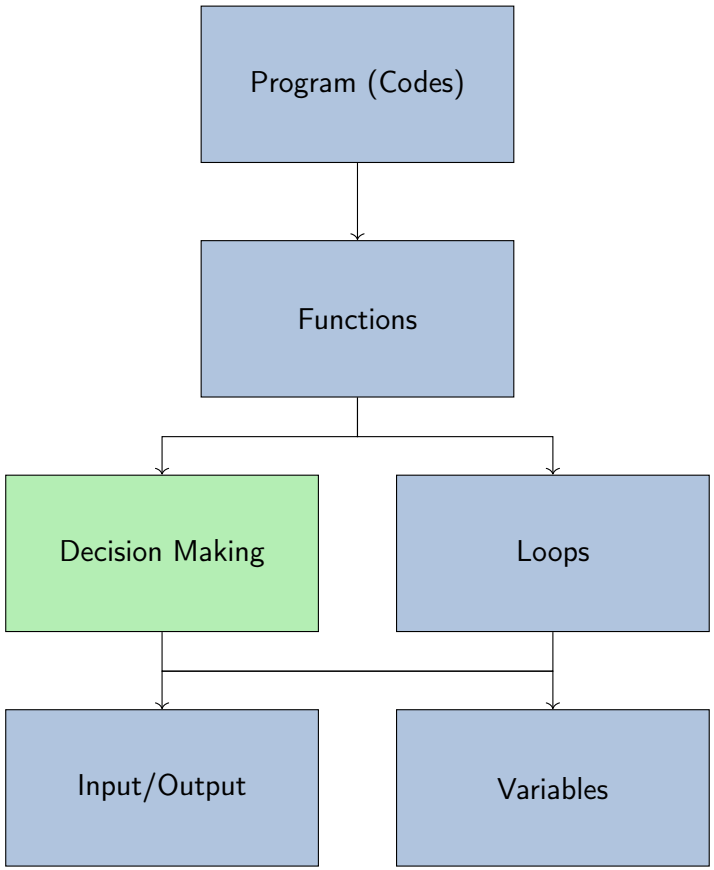
```
import random
num = random.randint(1, 10) # generates a random number
                            # We passed 1 and 10 into randint,
                            # so the number can only be
                            # from 1 to 10
print(num) # prints the number
```

Generating a random integer using `random` library

Another example:

```
import random
min = 15
max = 30
print(random.randint(min, max)) # prints a random number
                                # from 15 to 30
```

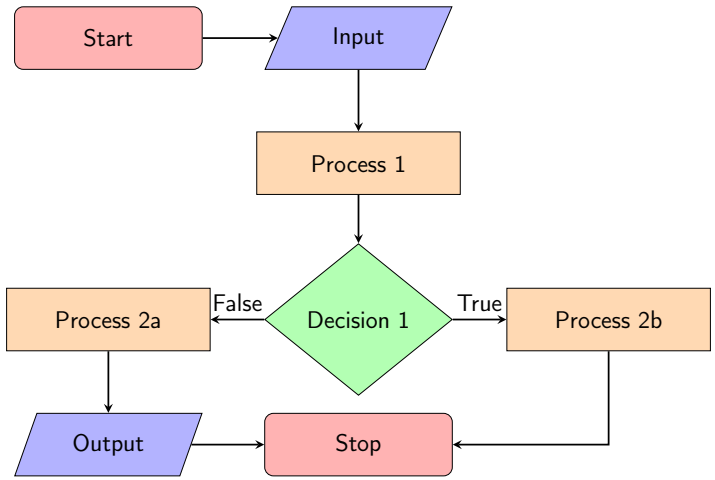
Contents


Program (Codes)
Functions
Decision Making
Loops
Input/Output
Variables

Decision Making

What is decision making?

We use condition(s) to decide whether some code should be run.



The if clause

```
a = 5 # a stores the integer 5
if a == 5:
    print("a stores 5.") # This line is activated

b = 10 # b stores the integer 10
if b == 5:
    print("b stores 5.") # This line is not activated
```

The if clause

If the condition is true, then the code under it is run.

The == operator

```
a = 5 # a stores the integer 5
if a == 5:
    print("a stores 5.") # This line is activated

b = 10 # b stores the integer 10
if b == 5:
    print("b stores 5.") # This line is not activated
```

The == operator

The operator == is used to compare 2 values. If the values on the both sides are the same, then it we consider it as true, false otherwise.

The if-else clause

```
a = 5 # a stores the integer 5
if a == 5:
    print("a stores 5.") # This line is activated
else:
    print("a does not store 5.")

b = 10 # b stores the integer 10
if b == 5:
    print("b stores 5.")
else:
    print("b does not store 5.") # This line is activated
```

The else statement

Code under the `else` statement is executed when the condition in `if` is not true.

The if-else clause

```
a = 5 # a stores the integer 5
if a == 5:
␣␣␣␣print("a stores 5.") # This line is activated
else:
␣␣␣␣print("a does not store 5.")

b = 10 # b stores the integer 10
if b == 5:
␣␣␣␣print("b stores 5.")
else:
␣␣␣␣print("b does not store 5.") # This line is activated
```

Indentation in Python

Indentation decides whether the code is under the if/else statements.
It does not have to be 4 spaces, but they have to be **consistent**.

The if-elif-else clause

```
a = 5 # a stores the integer 5
if a == 5:
    print("a stores 5.") # This line is activated
elif a == 10:
    print("a stores 10.")
else:
    print("a does not store 5 or 10.")
```

The elif statement

The elif (stands for else-if) statement is a secondary if statement that is run if the previous if/elif condition(s) are not true.

The if-elif-else clause

```
a = 15 # a stores the integer 15
if a == 5:
    print("a stores 5.")
elif a == 10:
    print("a stores 10.")
elif a == 15:
    print("a stores 15") # This line is activated
else:
    print("a does not store 5, 10 or 15.")
```

Stacking the elif statement

The elif statement can be stacked on top of one another.

Comparison Operators

We've learnt that == means "equal to". What are some other operators?

Operator	Meaning
==	equal to
>	larger than
>=	larger than or equal to
<	smaller than
<=	smaller than or equal to
!=	not equal to

Decision Making and Comparison Operators

```
a = 10 # a stores the integer 10
if a > 5:
    print("a is larger than 5")

if a >= 10:
    print("a is larger than or equal to 10")
```

In this example, both `print()` statements are activated.

Decision Making and Comparison Operators

```
a = 10 # a stores the integer 10
if a > 5:
    print("a is larger than 5")
elif a >= 10:
    print("a is larger than or equal to 10") # Not run
```

In this example, only the first `print()` statements are activated.

`if` vs `elif`

If a condition is fulfilled, any `elif` clauses afterwards will not be considered.

Logic Operators - and

The and operator denotes whether the 2 conditions are fulfilled **at the same time**.

Example:

```
a = 10 # a stores the integer 10
if a > 5 and a < 9:
    print("a is between 5 and 9")
else:
    print("a is not between 5 and 9") # This line is run
```

Logic Operators - or

The `or` operator denotes whether **any** of the 2 conditions are fulfilled.

Example:

```
a = 10 # a stores the integer 10
if a < 5 or a > 9:
    print("a is not between 5 and 9") # This line is run
else:
    print("a is between 5 and 9")
```

Logic Operators - not

The not operator reverses the condition.

Example:

```
a = 10 # a stores the integer 10
if not a == 5: # Same as a != 5
    print("a is not 5") # This line is run
else:
    print("a is 5")
```

Multiple Logic Operators

```
a = 10 # a stores the integer 10
if not a == 0 or a == 1: # Same as a != 0 or a == 1
    print("a is 0 or 1")
else:
    print("a is not 0 or 1")


b = 10 # b stores the integer 10
if b == 5 and not b == 5: # Impossible condition
    print("b is 5 and somehow not 5?")
else:
    print("Else statement")
```

Multiple Logic Operator (out of control)

We can use multiple logic operators together, but what about the rules?

```
a = 10 # a stores the integer 10
if not a == 0 and a == 1 or a == 3 and a * 2 == 10:
    print("What is going on in the conditions?")
else:
    print("Else statement")
```

Multiple Logic Operator (out of control)

We add brackets () to make our conditions clear.

```
a = 10 # a stores the integer 10
if (not a == 0 and a == 1) or (a == 3 and a * 2 == 10):
    print("Now the conditions are clearer")
else:
    print("Else statement")
```

Reminder

If you ever use > 1 and/or operators, add brackets to keep track of what your conditions are.

Summary

Variable types

There are 4 basic variable types: `int`, `bool`, `float` and `str`.

Arithmetic Operators

Some basic and commonly-used operators:

+:	add	-:	minus,
*:	multiply	/:	divide,
//:	quotient	%:	remainder,
**:	power		

Summary

The `print()` statement

```
print(*objects, sep=' ', end='\n', file=None, flush=False)
```

`*objects` - the things you want to print,
`sep` - the string that separates `objects` (when using commas),
`end` - the string to end the print statement with.
The other arguments can be ignored as they are rarely used.

The `input()` statement

```
input(prompt)
```

where `prompt` is quite literally what it means. It prints the prompt, then returns the value inputted as a string.

Summary

Type Conversion

To convert between types, you can simply surround the target with brackets, and call the type.

`int -> int()`; `bool -> bool()`; `float -> float()`; `str -> str()`.

`random.randint()`

`random.randint(a, b)`

`a` - the lower bound of your range

`b` - the upper bound of your range

This generates an integer `n` where $a \leq n \leq b$.

Summary

if, elif and else

if, elif and else clauses are used to decide whether some code should be executed. Whenever one is fulfilled, all others are ignored.

```
if condition1: # if condition1 is true
    # Do something, ignore all elif and else below

elif condition2: # if condition2 is true
    # Do something, ignore all elif and else below

elif condition3: # if condition3 is true
    # Do something, ignore all elif and else below

else: # if all the conditions above are false
    # Do something
```

Summary

Comparison Operators

There are 6 comparison operators:

Operator	Meaning
==	equal to
>	larger than
>=	larger than or equal to
<	smaller than
<=	smaller than or equal to
!=	not equal to

Summary

The `and` logic operator

The `and` operator makes it so that both conditions have to be fulfilled in order for the code it is under to execute.

The `or` logic operator

The `or` operator makes it so that only 1 of the conditions have to be fulfilled in order for the code it is under to execute.

The `not` logic operator

The `not` operator reverses the condition is it attached to.

Multiple logic operators

One can chain multiple logic operators together, but to be safe add brackets () to make sure the condition works as intended.

The end.

Made in LaTeX

Last updated: 18 Apr 2024