

# Coding in Python

## Python Basics - Part 1

IDPO 2910 Group 5

20 April 2024



THE HONG KONG  
UNIVERSITY OF SCIENCE  
AND TECHNOLOGY

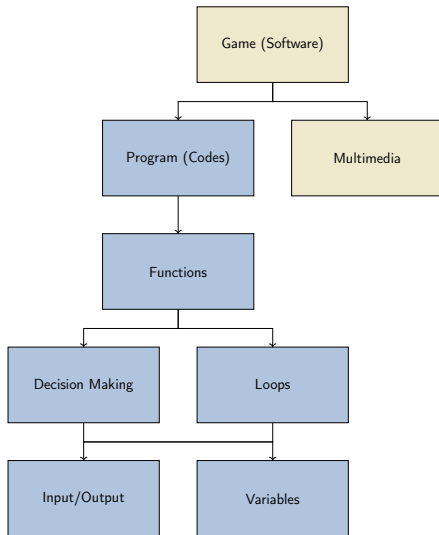
# What is Python?

Did you know? Python was made by someone who was bored.  
It's a language designed to be almost as understandable as English.  
You will be using Python 3. Why? Because Python 1 and 2 are too old.

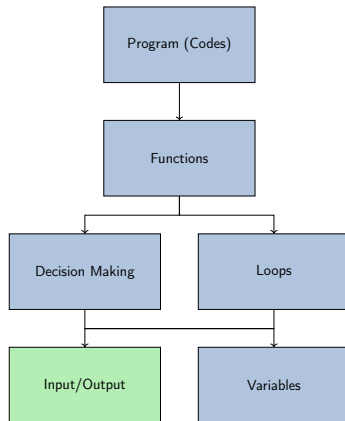


This is the logo of Python.

# World of Game Coding



# Contents



# The first thing in Python - print() function

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

# The first thing in Python - print() function

`print()` is a function that lets you print something, also known as text output.

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

Output:

```
>>> print("Word")
```

Word

```
>>> print("Haha hehe")
```

Haha hehe

# input() function

We know how to output strings, 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.

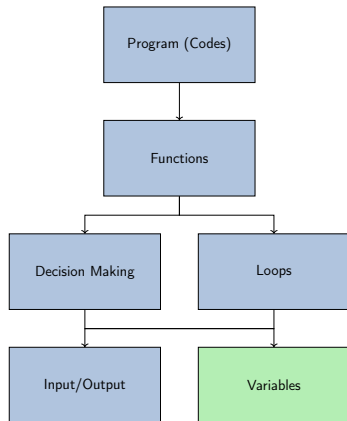
```
>>> input("Enter a number: ")  
Enter a number: 5
```

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

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



# Contents



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.

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.

```
a = 5           # Valid
b = 12          # Valid
c = 69420       # Valid
d = -1984       # Valid
e = 32.5        # This would become a float instead
f = '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 returns a float.

# Variables - Integer Arithmetic Operations

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 point(s).

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 0.0080000000000000002
```

## Inaccuracies

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

# Variables - Floats

What are floats?

Floats are numbers with decimal point(s).

We learnt about arithmetic operators: +, -, \*, /, //, %, \*\*

All of them, except // and % can be applied to floats.

```
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 0.0080000000000000002
```

## 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
```

That's it.

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

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
```

There are additional symbols in strings.

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

# Variables - Strings

```
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.

# Type conversion

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

Data Type	Command
Integer	<code>int()</code>
Float	<code>float()</code>
String	<code>str()</code>
Boolean	<code>bool()</code>

## Focus

We will focus on `int()` today as it will be needed in your game.



# Conversion with `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
```

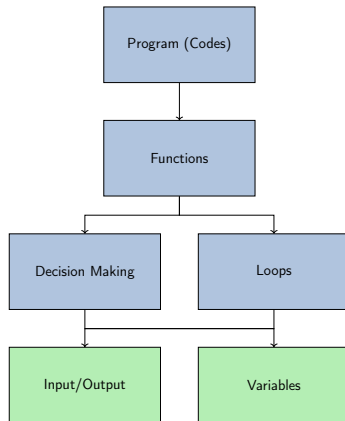
## Conversion with int()

```
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
```

# Contents



# Variables in output - Revisiting the print() function

How do we print variables?

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

## The comma

Using , in print() would add a space in between the 2 items.

# Variables in output - Revisiting 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.

# Variables in output - Revisiting 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!

# 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: ")
```

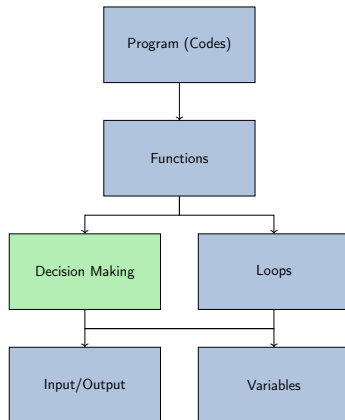
```
Enter your number: 50
```

```
>>> print(number)
```

```
50
```

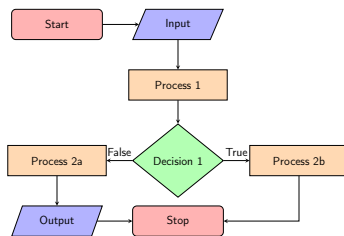
```
>>> print(int(number) + 1000) # 1050
```

# Contents



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 becomes **True**. It becomes **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
<code>==</code>	equal to
<code>&gt;</code>	larger than
<code>&gt;=</code>	larger than or equal to
<code>&lt;</code>	smaller than
<code>&lt;=</code>	smaller than or equal to
<code>!=</code>	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
```

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 % 2 != 0 or a == 1: # Same as a % 2 == 0 or a == 1
    print("a is even or equal to 1")
else:
    print("a is odd and not equal to 1")

b = 10 # b stores the integer 10
if b == 5 and not b % 2 != 0: # Impossible condition
    print("b is 5 and somehow even?")
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 == 1:
    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 == 1):
    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.

# 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
```

# Generating a random integer using random library

Another example with decision making:

```
import random
num = random.randint(1, 10)
if num < 5:
    print("The number is smaller than 5")
else:
    print("The number is larger than or equal to 5")
```

# Summary

## Variable types

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

## Arithmetic Operators

Some basic and commonly-used operators:

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

## 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().`



# 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 output, then returns the value inputted as a string.

# 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

## 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 it is 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.

# Summary

```
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  $a \leq n \leq b$ .

The end  
Written in  $\text{\LaTeX}$   
Last updated: 18 Mar 2024