Introduction to Python

What is Python?

- Python is a widely-used, interpreted, object-oriented, and high-level programming language with dynamic semantics, used for general-purpose programming.

Install Python and pycharm

- https://www.python.org/downloads/

- **PyCharm - Python IDE for Professional Developers**
- https://www.jetbrains.com/pycharm/download/#section=windows

Hello, World!

- `print("Hello, World!“)`
- **Output:**

```
Console >_

Hello, World!
```

As you can see, the first program consists of the following parts:
•the word `print`;
•an opening parenthesis;
•a quotation mark;
•a line of text: `Hello, World!`;
•another quotation mark;
•a closing parenthesis.

Print function

- **The print() function**
- As we said before, a function may have:
  1. an **effect**;
  2. a **result**.
  3. There's also a third, very important, function component - the **argument**(s).
- Python functions, may accept any number of arguments - as many as necessary to perform their tasks.
- Note: **any number includes zero** - some Python functions don't need any argument.

```
print("Hello, World!“)
```

Variables

- Variables is boxes" (containers) store value.
- What does every Python variable have?
  1. a name;
  2. a value (the content of the container)
- If you want to give a name to a variable, you must follow some strict rules:
  - the name of the variable must be composed of upper-case or lower-case letters, digits, and the character _ (underscore)
  - the name of the variable must begin with a letter;
  - the underscore character is a letter;
  - upper- and lower-case letters are treated as different
  - the name of the variable must not be any of Python's reserved words (the keywords - we'll explain more about this soon).

Creating variables

- var = 1
- print(var)

```
Console >_
1
```

- var = 1
- account_balance = 1000.0
- client_name = 'John Doe'
- print(var, account_balance, client_name)
- print(var)

```
Console >_
1 1000.0 John Doe
1
```

Comments

- Comment is a remark inserted into the program, which is **omitted at runtime**.
- Created for human not for python
- In Python, a comment is a piece of text that begins with a # (hash) sign
- `# This program evaluates the hypotenuse c.`
- `# a and b are the lengths of the legs.`
- `a = 3.0`
- `b = 4.0`
- `c = (a ** 2 + b ** 2) ** 0.5 # We use ** instead of square root.`
- `print("c =", c)`
-

Comments

- Comments may be useful in another respect - you can use them to **mark a piece of code that currently isn't needed.**

```
# This is a test program.
x = 1
y = 2
# y = y + x
print(x + y)
```

Console >_

3

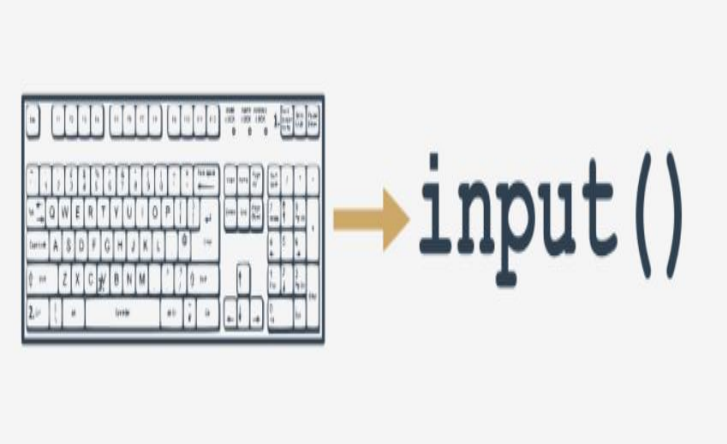
- If you'd like to quickly comment or uncomment multiple lines of code, select the line(s) you wish to modify and use the following keyboard shortcut: **CTRL** + **/**

The input() function

- The input() function is able to read data entered by the user and to return the same data to the running program.

```
print("Tell me anything...")
anything = input()
print("Hmm...", anything, "... Really?")
```

-



The input() function with an argument

```
anything = input("Tell me anything...")
print("Hmm...", anything, "...Really?")
```

- the input() function is invoked with one argument - it's a string containing a message;
- the message will be displayed on the console before the user is given an opportunity to enter anything;
- input() will then do its job
- the result of the input() function is a string.
- This means that **you mustn't use it as an argument of any arithmetic operation**, e.g., you can't use this data to square it, divide it by anything, or divide anything by it.

```python
# Testing TypeError message.
anything = input("Enter a number: ")
something = anything ** 2.0
print(anything, "to the power of 2 is", something)
```

Console >_

```
Enter a number: 123
Traceback (most recent call last):
  File "main.py", line 4, in <module>
    something = anything ** 2.0
TypeError: unsupported operand type(s) for ** or pow(): 'str' and 'float'
```

Type casting

- Python offers two simple functions to specify a type of data and solve this problem - here they are: int() and float().
- the int() function **takes one argument** (e.g., a string: int(string)) and tries to convert it into an integer; if it fails, the whole program will fail too (there is a workaround for this situation, but we'll show you this a little later);
- the float() function takes one argument (e.g., a string: float(string)) and tries to convert it into a float (the rest is the same).

```
anything = float(input("Enter a number: "))
something = anything ** 2.0
print(anything, "to the power of 2 is", something)
```

```
Console >_
Enter a number: 12
12.0 to the power of 2 is 144.0
```

More about input() and type casting

```
leg_a = float(input("Input first leg length: "))
leg_b = float(input("Input second leg length: "))
hypo = (leg_a**2 + leg_b**2) ** .5
print("Hypotenuse length is", hypo)
```

```
Console >_

Input first leg length: 3
Input second leg length: 5
Hypotenuse length is 5.830951894845301
```

String operators - introduction

- **Concatenation**

The + (plus) sign, when applied to two strings, becomes a concatenation operator: string + string

```
fnam = input("May I have your first name, please? ")
lnam = input("May I have your last name, please? ")
print("Thank you.")
print("\nYour name is " + fnam + " " + lnam + ".")
```

```
Console >_

May I have your first name, please? SHREEN
May I have your last name, please? KHALAF
Thank you.

Your name is SHREEN KHALAF.
```

Replication

- The * (asterisk) sign, when applied to a string and number (or a number and string, as it remains commutative in this position) becomes a replication operator:
- "James" * 3 gives "JamesJamesJames"
- 3 * "an" gives "ananan"
- 5 * "2" (or "2" * 5) gives "22222" (not 10!)
-

```
print("+" + 10 * "-" + "+")
print( ("|" + " " * 10 + "|\n") * 5, end="")
print("+" + 10 * "-" + "+")
```

Console >_

```
+----------+
|          |
|          |
|          |
|          |
|          |
+----------+
```

Str function

- Type conversion: str(): convert a number into a string

```
leg_a = float(input("Input first leg length: "))
leg_b = float(input("Input second leg length: "))
print("Hypotenuse length is " + str((leg_a**2 + leg_b**2) **
.5))
```

Conditions and conditional execution

- The first form of a conditional statement,

```
if true_or_not:
    do_this_if_true
```

- This conditional statement consists of the following, strictly necessary, elements in this and this order only:

1. the if keyword;
2. one or more white spaces;
3. an expression (a question or an answer) whose value will be interpreted solely in terms of True (when its value is non-zero) and False (when it is equal to zero);
4. a colon followed by a newline;
5. an indented instruction or set of instructions (at least one instruction is absolutely required); the indentation may be achieved in two ways - by inserting a particular number of spaces (the recommendation is to use **four spaces** of indentation), or by using the **tab** character; note:

Conditional execution: the if-else statement

```
if true_or_false_condition:
    perform_if_condition_true
else:
    perform_if_condition_false
```

How to identify the larger of two numbers:

```python
# Read two numbers
number1 = int(input("Enter the first number: "))
number2 = int(input("Enter the second number: "))

# Choose the larger number
if number1 > number2:
    larger_number = number1
else:
    larger_number = number2

# Print the result
print("The larger number is:", larger_number)
```

Console >_

```
Enter the first number: 12
Enter the second number: 15
The larger number is: 15
```

Nested if-else statements

```
 if the_weather_is_good:
    if nice_restaurant_is_found:
        have_lunch()
    else:
        eat_a_sandwich()
else:
    if tickets_are_available:
        go_to_the_theater()
    else:
        go_shopping()
```

```python
x = 10

if x > 5:  # True
    if x == 6:  # False
        print("nested: x == 6")
    else:
        print("nested: else")
else:
    print("else")
```

The elif statement

- **The elif statement:** The second special case introduces another new Python keyword: **elif**. As you probably suspect, it's a shorter form of **else if**.
- **elif** is used to check more than just one condition, and to stop when the first statement which is true is found.

```
if the_weather_is_good:
    go_for_a_walk()
elif tickets_are_available:
    go_to_the_theater()
elif table_is_available:
    go_for_lunch()
else:
    play_chess_at_home()
```

elif example

```
 x = int(input("Enter the value of x: "))
if x == 10: # True
print("x == 10")
if x > 15: # False
print("x > 15")
elif x > 10: # False
print("x > 10")
elif x > 5: # True
print("x > 5")
else:
print("else will not be executed")
```

Loops

Looping your code with while

- while conditional_expression:

  instruction
- **while** repeats the execution as long as the condition evaluates to True
- while conditional_expression:

  instruction_one

  instruction_two

  instruction_three

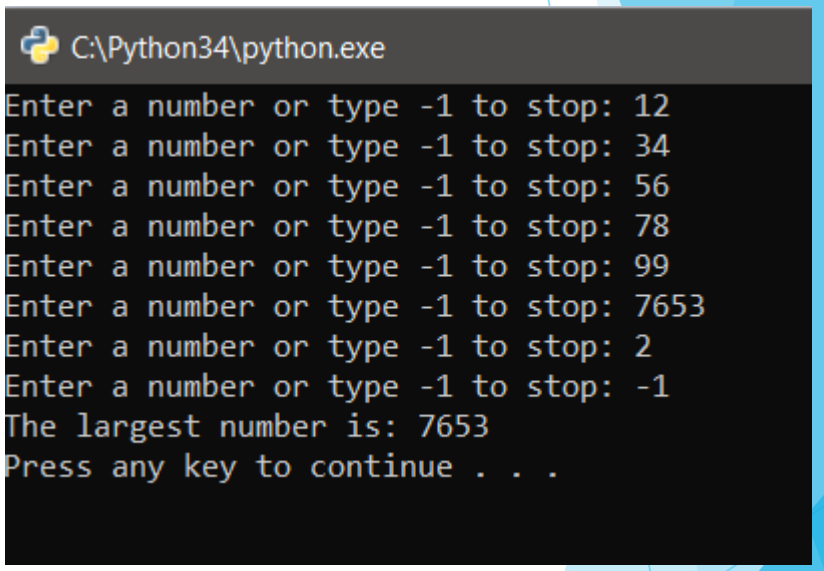
  :

  :

  instruction_n

Get largest number Example

```
# Store the current largest number here.
largest_number = -999999999
# Input the first value.
number = int(input("Enter a number or type -1 to stop: "))
# If the number is not equal to -1, continue.
while number != -1:
# Is number larger than largest_number?
    if number > largest_number:
        # Yes, update largest_number.
        largest_number = number
        # Input the next number.
    number = int(input("Enter a number or type -1 to stop: "))
# Print the largest number.
print("The largest number is:", largest_number)
```



Looping your code with for

- for i in range(100):
  - # do_something() pass
- the for keyword opens the for loop;
- any variable after the for keyword is the control variable of the loop; it counts the loop's turns, and does it automatically;
- the in keyword introduces a syntax element describing the range of possible values being assigned to the control variable;
- the **range**() function (this is a very special function) is responsible for generating all the desired values of the control variable; in our example, the function will create (we can even say that it will feed the loop with) subsequent values from the following set: **0**, 1, 2 .. 97, 98, **99;** note: in this case, the range() function starts its job from 0 and finishes it one step (one integer number) before the value of its argument;

Looping your code with for

```
for i in range(10):
    print("The value of i is currently", i)
```

Console >_

```
The value of i is currently 0
The value of i is currently 1
The value of i is currently 2
The value of i is currently 3
The value of i is currently 4
The value of i is currently 5
The value of i is currently 6
The value of i is currently 7
The value of i is currently 8
The value of i is currently 9
```

Looping your code with for

- The range() function invocation may be equipped

with two arguments, not just one:

```
for i in range(2, 8):
    print("The value of i is currently", i)
```

```
Console >_

The value of i is currently 2
The value of i is currently 3
The value of i is currently 4
The value of i is currently 5
The value of i is currently 6
The value of i is currently 7
```

- In this case, the first argument determines the **initial** (first) value of the control variable.
- The last argument shows the first value the control variable will not be assigned.
- Note: the range() function accepts only **integers** as its arguments, and generates sequences of integers.

More about the for loop and the range() function with three arguments

- The range() function may also accept three arguments - take a look at the code in the editor.
- The third argument is an **increment** - it's a value added to control the variable at every loop turn (as you may suspect, the **default value of the increment is 1**).
- for i in range(2, 8, 3):
  - print("The value of i is currently", i)

```
Console >_
The value of i is currently 2
The value of i is currently 5
```

Function

Your first function

```
def my_function():
    # function body
def message():
    print("Enter a value: ")
```

function's invocation

- message()

```
def message():
    print("Enter a value: ")

print("We start here.")
message()
print("We end here.")
```

Console >_

```
We start here.
Enter a value:
We end here.
```

- **You mustn't invoke a function which is not known at the moment of invocation.**
- `print("We start here.")`
- `message()`
- `print("We end here.")`
- `def message():`
- `print("Enter a value: ")`

```
Console >_

We start here.
Traceback (most recent call last):
  File "main.py", line 2, in <module>
    message()
NameError: name 'message' is not defined
```

- **You mustn't have a function and a variable of the same name.**
- def message():
- print("Enter a value: ")
- message = 1
- print(message)

```
Console >_

1
```

- Assigning a value to the name message causes Python to forget its previous role. The function named message becomes unavailable.

Function with arguments

```
def hello(name):    # defining a function
    print("Hello,", name)    # body of the function

name = input("Enter your name: ")
hello(name)    # calling the function
```

Positional parameter passing

- A technique which assigns the ith (first, second, and so on) argument to the ith (first, second, and so on) function parameter is called **positional parameter passing**, while arguments passed in this way are named **positional arguments**.

```
def introduction(first_name, last_name):
    print("Hello, my name is", first_name, last_name)
introduction("Luke", "Skywalker")
introduction("Jesse", "Quick")
introduction("Skywalker", "Luke")
introduction("Skywalker", "Luke")
introduction("Quick", "Jesse")
```

Console >_

```
Hello, my name is Luke Skywalker
Hello, my name is Jesse Quick
Hello, my name is Skywalker Luke
Hello, my name is Quick Jesse
```

- Can you make the function more culture-independent?

Keyword argument passing

```
def introduction(first_name, last_name):
    print("Hello, my name is", first_name, last_name)

introduction(first_name = "James", last_name = "Bond")
introduction(last_name = "Skywalker", first_name = "Luke")
```

```
Console >_
Hello, my name is James Bond
Hello, my name is Luke Skywalker
```

- The concept is clear - the values passed to the parameters are preceded by the target parameters' names, followed by the = sign.

Effects and results: the return instruction

- To get functions to return a value (but not only for this purpose) you use the return instruction.
- The return instruction has two different variants - let's consider them separately.
- **return without an expression**
- The first consists of the keyword itself, without anything following it.
- `def happy_new_year(wishes = True):`
- `print("Three...")`
- `print("Two...")`
- `print("One...")`
- `if not wishes:`
- `return`
- `print("Happy New Year!")`
- happy_new_year()
-

- **return with an expression**
- `def boring_function():`
- `return 123`
- `x = boring_function()`
- `print("The boring_function has returned its result. It's:", x)`

```
The boring_function has returned its result. It's: 123
```

A few words about None: continued

```
def strange_function(n):
    if(n % 2 == 0):
        return True

print(strange_function(2))
print(strange_function(1))
```

Console >_

```
True
None
```