Introduction to programming with Python

Helpful Tips & Conventions

Helpful tips for following these lessons and conventions being adopted

helpful tips

DO NOT copy/paste code from slides → it may lead to error (mostly indentation and artifacts errors) and also writing will help you grasp the syntax faster (when copy/pasting is necessary I will put the content into a Jupyter Notebook)

If this Icon is present in the slide it means that there are some resources (code, examples, exercises) on the Jupyter Notebook relative to this lesson

If this Icon is present it means that there is a external resource reference in the slide

Introduction to programming

programming is . . .

Programming is the process of giving instructions to a computer to perform specific tasks or solve a specific problem.

The process of solving a problem begins with detailed specification of the problem itself.

We can then suppose what methods that we know could solve the problem.

We create an outline of the possible solution, written in human language: **pseudocode**

Pseudocode than can be translate into computer language: script, source code, or more generally computer program.

Then a compiler (or interpreter) transform (or translate) the source code into machine language.

exercise - guess a number (I)

A game where the player should guess which number I've chosen in the range 1-20. The program must say if I won or if the guess is higher or lower. The player have 5 chances at most.

What should I do?

- divide into smaller problem than I know how to solve
- write pseudo code

exercise - guess a number (II)

A game where the player should guess which number I've chosen.

1. The computer select a number
2. The computer asks the player to guess
3. The player types a number on the keyboard and the computer reads it in
4. The computer compares the input number against the chosen one and if the two match, then the player wins. Otherwise, the computer wins

exercise - guess a number (III)

A game where the player should guess which number I've chosen in the range 1-20.

1. The computer select a number in the range 1-20
2. The computer asks the player to guess
3. The player types a number on the keyboard and the computer reads it in.
    3.1 The computer tell the player to try again if the guess is out of 1-20 range
4. The computer compares the input number against the chosen one and if the two match, then the player wins. Otherwise, the computer wins.

exercise - guess a number (IV)

A game where the player should guess which number I've chosen in the range 1-20. The program must say if I won or if the guess is higher or lower.

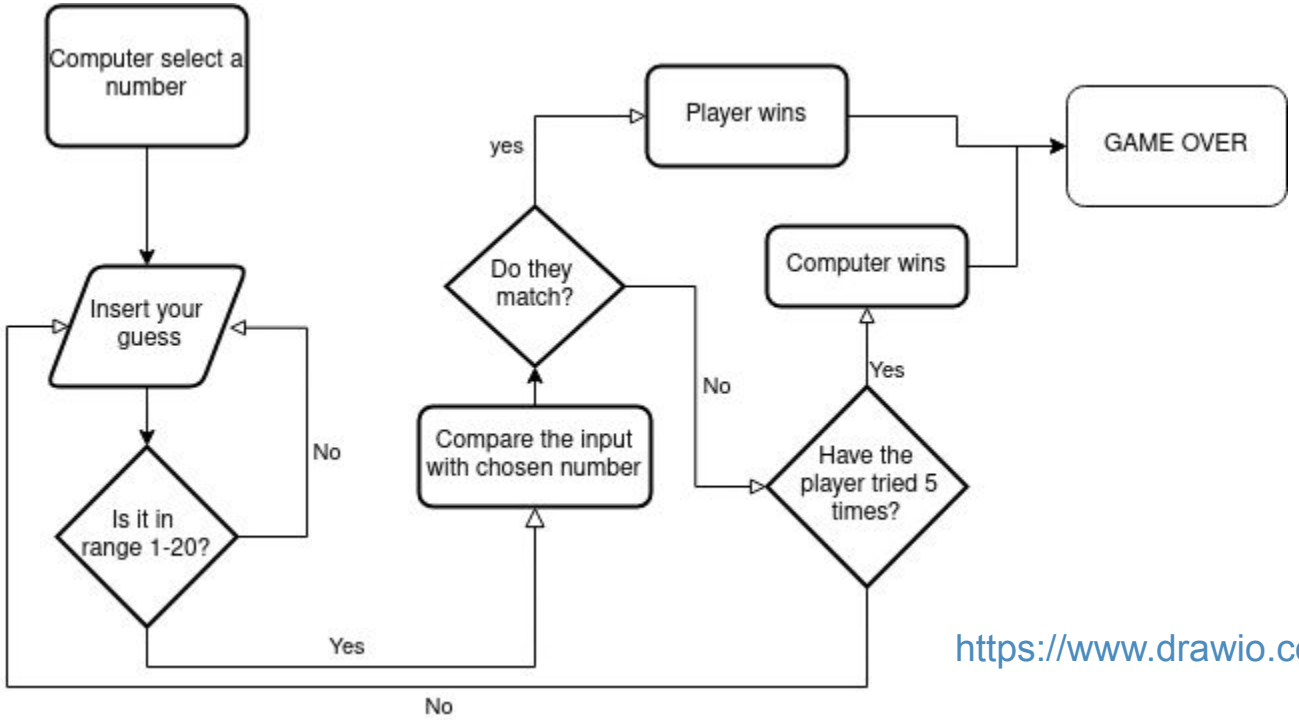
1. The computer select a number in the range 1-20
2. The computer asks the player to guess
3. The player types a number on the keyboard and the computer reads it in.
    3.1 The computer tell the player to try again if the guess is out of 1-20 range
4. The computer compares the input number against the chosen one
    4.1. If the two match, then the player wins. Go to Step 6
    4.2. Otherwise, ~~the computer wins.~~
        4.2.1 If the guess is higher tell the player to try with a lower number
        4.2.2 If the guess is lower tell the player to try with a higher number
5. Repeat from Step 2
6. Game over

exercise - guess a number (V)

A game where the player should guess which number I've chosen in the range 1-20. The program must say if I won or if the guess is higher or lower. The player have 5 chances at most.

1. The computer select a number in the range 1-20
2. The computer asks the player to guess
3. The player types a number on the keyboard and the computer reads it in.
    3.1 The computer tell the player to try again if the guess is out of 1-20 range
4. The computer compares the input number against the chosen one
    4.1. If the two match, then the player wins. Go to Step 6
    4.2. Otherwise, ~~the computer wins.~~
        4.2.1 If already tried 5 times the player lose. Go to step 6
        4.2.2 Otherwise increase number of tries and
            4.2.2.1 If the guess is higher tell the player to try with a lower number
            4.2.2.2 If the guess is lower tell the player to try with a higher number
5. Repeat from Step 2
6. Game over

exercise - guess a number (VI)



https://www.drawio.com/

programming principles

Three main principles

Sequentiality: Executing instructions in order, one after the other

Condition: Making decisions based on specific conditions or criteria

Iteration: Repeating a set of instructions multiple times

introduction to python

python is

Python is an **open-source**, **high-level**, **interpreted**, ***general-purpose*** programming language. It is **dynamically typed** and **multi paradigms**

...meaning what?
high-level → a language with **strong abstraction** from machine details (more similar to english than to machine language);

interpreted → an **interpreter** execute commands/programs directly without needing the *translation* of the whole program;

general-purpose → allows to build software in a wide variety of application domains;

multiple programming paradigms → support different programming paradigms *(procedural, object-oriented, functional among others)*

open-source → code is public and editable;

dynamically typed → verify the type safety of a program at runtime

history

Guido van Rossum (BDFL - Benevolent dictator for life) is the creator of Python.

Python was born as a successor of ABC language, a middle 80' general-purpose code language.

Name is inspired by "Monty Python's Flying Circus" [van Rossum was passionate]

"older" than Java → Java was born in 1995;

version 2 → released in 2000 (EOL expected in 2015 extended to 2020)

version 3 → released in 2008 (not backwards compatible)

philosophy → summarised in *The Zen of Python* [`import this`]

interpreters

Cpython

CPython main implementation of Python.
Written in C and Python,
CPython is the most used implementation of the language.

Jython

Jython, successor of JPython, is an implementation of the Python language written in Java.
Jython programs can import and use Java classes.

Pypy

PyPy is an alternative implementation of Cpython.
PyPy can be faster than CPython, because PyPy is a just-in-time compiler, while CPython is an interpreter.

Stackless Python

Stackless Python is a Python interpreted, so called because it doesn't depends on C stack for its stack.
The most important feature are the microthread, that avoid the usual load a OS threads.

interpreters

MicroPython

Ad-hoc implementation of Python 3 for microcontroller (i.e. arduino and esp board).
It doesn't allow the whole py3 standard library but adds some modules for hardware control.

Anaconda

Great success in Data Science.
Adds to standard library a great load of modules for data science
(i.e. numpy e pandas)

Ipython

Ipython boost CPython making it more interactive.
Great interaction with command shell.
The base for Jupyter Notebook project.

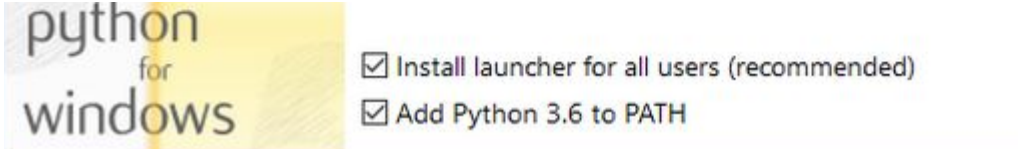
install python interpreter

install python interpreter

Python official documentation at: https://wiki.python.org

installing on linux system → https://wiki.python.org/moin/BeginnersGuide/Download
arch -> already installed
centos/redhat/fedora → yum python3 python3-devel
debian/ubuntu → apt install python3.x python3.x-dev
installing on MacOs → https://installpython3.com/mac/
installing on WIndows → download .exe from https://www.python.org/downloads/

WARNING: remember to check the "Add Python 3.X to PATH" checkbox during installation on Windows machine



is it working?

verify if working:

open terminal (powershell in windows case) and exec → python

you should obtain:

```
[user@hostname ~]$ python
Python 3.11.5 (main, Sep  2 2023, 14:16:33) [GCC 13.2.1 20230801] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

IDE

Integrated Development Environment → a software that helps programmers in writing software

Installing pycharm → https://www.jetbrains.com/pycharm/download/

Installing vscode → https://code.visualstudio.com/Download

Jupyter Notebook → an application (di tipo server-client che possiamo utilizzare sui nostri desktop) that allow to create documents with an integrated python "kernel", that is an internal interpreted that allow to execute python code

Installing Jupyter → (command prompt) `pip install jupyterlab` (or, if it doesn't work: `python -m pip install jupyterlab`)
Execute Jupyter → (command prompt) `jupyter`

documentation → https://jupyter.org/install.html

Why Jupyter?

An easy and flexible interface that allow you to take notes while executing python code. Moreover, it is an "isolated" way to use some python library like *numpy* or *pandas* without having to install *anaconda* in your system (*anaconda* is not recommended on linux system).

the python language

lexical structure

The end of the line determine the end of a statement. If a statement is longer than 80 characters I can use backslash (\) to continue into a new line.
Opening parenthesis (([{) determine the end of the statement to be at their closing (no need for backslash, newlines will be ignored).

Python uses indentation to express the structure of program blocks. So, a block is a set of consecutive lines, all indented by the same amount.

In indentation, it's best not to mix spaces and tabs. It's recommended to use 4 spaces, IDEs usually automatically convert tabs to 4 spaces.

charset - tokens

In Python 3 source code is encoded into UTF-8.
Encoding is a typical problem when working in windows environment, where default encoding is iso-8859-1.

In python every statement is a sequence of lexical components also known as *tokens*, these are *identifiers, keywords, operators, delimiters, literals.*

identifiers → a name used to specify a variable, function, class, module or any other object. It starts with a letter or an underscore(_) followed by 0 or more letters, underscores or numbers. Case matter!

keywords → reserved identifiers used in language syntax (and, as, assert, break, class, continue, def, del, elif, else, False, finally, for, from, global, if, import, in, is, lambda, None, nonlocal, not, or, pass, raise, return, True, try, while, with, yield)

tokens

operators → + - * / = % ** // << >> & | ^ ~ < <= > >= <> != == += -= *= /= //= %= &= |= ^= >>= <<= **=

delimiters → ( ) [ ] { } , : . ` ; @
' and " surrounds literals of string type
outside of a string begins a comment (a non executing part of the program)
\ at the end of a line make so the following line constitute a single logical line with the current

literals → In a program, it involves the representation of data (number, string, container)

```
>>> 42 # Integer literal
42
>>> 1.0j # Imaginary literal
1j
>>> """ Good
... night""" # Triple-quoted string literal
'Good\nnight'
>>> [1, 2, 3] # list
[1, 2, 3]
```

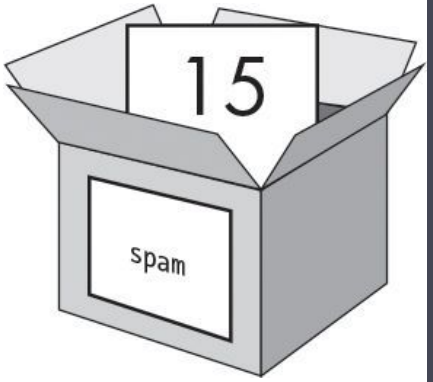
use the python command prompt

```
[user@hostname ~]$ python
Python 3.11.5 (main, Sep  2 2023, 14:16:33) [GCC 13.2.1 20230801] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> 2+2+2+2+2
10
>>> 8*6
48
>>> 10-5+6
11
>>> 2  +                    2
4
>>>8*3/2+2+7-9
12.0
```

variables (I)

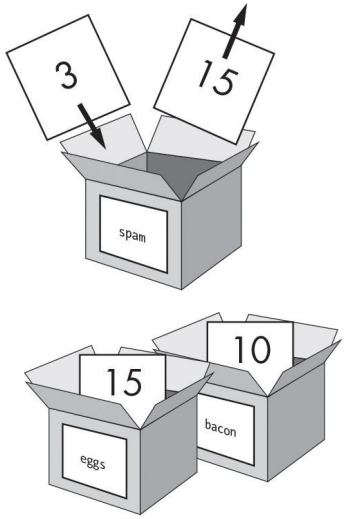
A variable is a name the programmer use to represent a value.

It represents a *box* where the computer store that value.



https://inventwithpython.com/invent4thed/chapter1.html

```
>>> spam = 15
>>> spam
15
>>> spam + 5
20
>>> spam
15
>>> spam = 3
>>> spam
3
>>> spam = spam + 5
>>> spam
8
```




```
>>> eggs = 15
>>> bacon = 10
>>> spam = bacon +
eggs
>>> spam
25
>>> eggs
15
>>> bacon
10
```

variables (II)

In Python, text values are called strings.

You can combine string values with operators to make expressions, just as you did with integer and float values.

```
>>> spam = 'hello'
>>> spam
'hello'
>>> 'Hello' + 'World!'
'HelloWorld!'
>>> spam + ' World!'
'hello World!'
```

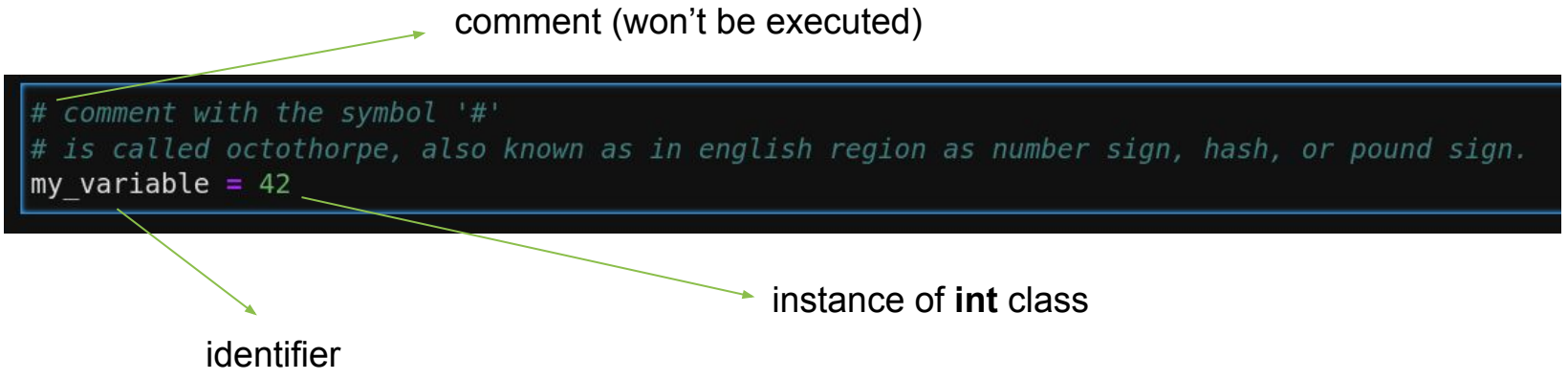
Guess The Number

Let's try to read our first program

data types

python

Python is an object oriented language with dynamic typing of variables (though these are strongly typed). Not necessary to declare the variable before using it, nor declaring its type. Each variable is an instance of a class being assigned to an identifier.



python - numbers -

Main types of numbers that Python support are integers and floating point numbers (it supports also complex numbers). [The built-in function type() return the object type]

<class 'int'> <class 'float'>

```
>>> type(3)
<class 'int'>
>>> x = 3
>>> type(x)
<class 'int'>
>>> y = int('3')
>>> type(y)
<class 'int'>
>>>
```

check instance

check instance assigned to an identifier

check instance created with constructor

```
>>> type(1.5)
<class 'float'>
>>> x = 1.5
>>> type(x)
<class 'float'>
>>> y = float('1.5')
>>> type(y)
<class 'float'>
>>>
```

python - numeri -

The interpreter behaves like a simple calculator: you can write an expression, and the interpreter will return its value. The syntax of the expression is simple: the operators +, -, *, and / work exactly as in most other languages, and parentheses () can be used for grouping.

```
>>> # python respect conventional order of operations (PEMDAS)
>>> 2 + 2
>>> 50 - 5 * 6
20
>>> (50 - 5 * 6) / 6
3.33333333333333335
>>> 4 / 2 # division return always a float
2.0
>>>
```

python - strings -

Strings can be defined interchangeably with single quotes (' ') or double quotes (" "). They are immutable sequences of textual characters that can be iterated over.

```
>>>  "hello" == 'hello' # single or double quotes doesn't make any difference
True
>>> print('hello') # built-in funtion print()
hello
>>>
```

```
>>> for letter in 'hello':
...         print(letter)
...
h
e
l
l
o
>>>
```

iterazione su string con *for* loop

python - None/bool -

None is the sole instance of the NoneType class; it can be used as a null value to assign to an identifier (in other languages, the keyword null is used), and it is the default return value of functions.

```
>>> x = None
>>> type(x)
<class 'NoneType'>
>>> def no_return():
...       pass
...
>>> print(no_return())
None
>>>
```

The bool class is used to manipulate logical (boolean) values, and the only two instances of this class are True and False. True is equivalent to 1, and False is equivalent to 0.

```
>>> type(True)
<class 'bool'>
>>> type(False)
<class 'bool'>
>>> True == 1 and False
== 0
True
>>> (True + 3) * False
0
>>>
```

print

python - stringhe -

The print() function allows you to output objects (typically to the screen but also to files or other streams). The objects passed into the function are converted into strings and written to the stream. You can include mathematical expressions, variables, and more within print, and the result will be printed.

```
>>> print("Python print function")
Python print function
>>> print(3 + 7)
10
>>> a = 'Python string'
>>> a
'Python string'
>>> print(a)
Python string
>>>
```

print a string

print algebraic expression result

print variable value

python - strings -

Print has many parameters. Among the most commonly used, end must certainly be mentioned.

end → default value is '\n' (escape sequence: new line). Can be replaced with any string.

end=default('\n')

end='' (avoid newline after the string)

Third function printed on the same line as the previous one (using the escape sequence: horizontal tab).

no object: print only a newline

INPUT

```
print("first line")
print("second line", end='')
print("\tsame line", end='\ttab!\n')
print()
print("third line", end='\t...end of the third line\n')
```

OUTPUT

```
first line
second line     same line       tab!

third line      ...end of the third line
```

slicing

python - slicing -

Python provides a notation called slicing (to slice = to cut) that allows you to access a series of elements in an ordered sequence (string, list, etc.) using their index. The first element corresponds to index 0, the second to 1, and so on.

The syntax uses square brackets with the following parameters:
[{start}: stop {: step}]

- **start** → starting index [optional]. 0 is the default start index
i.e. "home"[2:] == "me"
- **stop** → end index NOT included. If not specified, slicing foes to the end of the object
i.e. "home"[:3] == "ho"
- **step** → Index difference between one element and the next [optional]. Default is 1 (all elements between start and stop [exclusive]).
i.e. "abcdefg"[0:5:2] == "ace"

python - slicing -

Examples:

```
>>> s = 'Slicing test'
```

```
>>> s[0]
'S'
>>> s[1]
'l'
>>> s[:4]
'Slic'
>>>
```

Location index of a single element

Slicing from the first till the element with index 4 (not included)

Location index of a single element (negative index for going backward)

Slicing all element with even index

Negative steps to obtain the elements in backwards order

```
>>> s[-1]
't'
>>> s[ : : 2]
'Siigts'
>>> s[ : : -1]
'tset gnicilS'
>>>
```

operators

python - operators -

- Operators are used to perform operations on values and variables.
- Operators can manipulate individual objects and return a result.
- The elements involved are referred to as operands or arguments.
- Operators are represented by keywords or special characters.

python - arithmetic operators -

Arithmetic operators are used with numerical values to perform common mathematical operations.

Operator	Common name	Example
+	Addition	3 + 2 = 5
-	Subtraction	3 - 2 = 1
*	Multiplication	3 * 2 = 6
/	Division	3 / 2 = 1.5
%	Module (return the remainder)	3 % 2 = 1
//	Integer division (floor division)	3 // 2 = 1
**	Power	3 ** 2 = 9

python - augmented assignment operators -

Augmented assignment operators are used to assign values to identifiers.

Operator	Augmented assignment	Standard version
=	x = 5	x = 5
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5
//=	x //= 5	x = x // 5
**=	x **= 5	x = x ** 5

python - comparison operators -

Comparison operators are used to compare two values.

Operator	Common name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Lower than	x < y
>=	Greater or equal than	x >= y
<=	Lower or equal than	x <= y

python - comparison operators -

Unlike other languages, Python offers the possibility to chain comparison operators:

Let's try to implement this condition:

a < b < c

The most common implementation is:

a < b and b < c

Python allow to write:

a < b < c

```
>>> 2 < 5 and 5 < 7
True
>>> 2 < 5 < 7
True
>>> 2 < 5 < 5
False
>>>
```

python - logical operators -

Logical operators are used to combine conditional statements.

Operator	Description	Example
and	Return True is both condition are True	x < 5 and x < 10
or	Return True if at least one of the condition is True	x < 5 or x < 4
not	Invert the result, return False if condition is True and viceversa	not x > 5

python - identity operators -

Identity operators are used to compare objects, not to check if the values are equal, but whether they are the same object (occupying the same memory location).

Operator	Description	Example
is	Return True only if object being compared are EXACTLY the same object.	x is y
is not	Return True only if operators being compared ARE NOT the same object.	x is not y

python - operatori di appartenenza -

Membership operators check the membership of an object in a sequence such as a list, string, or tuple.

Operator	Description	Esempio
in	Returns True if the first operand is present in the sequence represented by the second operand.	x in y
not in	Returns True if the first operand is NOT present in the sequence represented by the second operand.	x not in y

conditional statements

python - conditional statement -

```
if first_condition:
    first_block
elif second_condition:
    second_block
elif third_condition:
    third_block
else:
    fouth_block
```

```
def f(n):
    if n == 1:
        return 1
    else:
        return f(n-1)
```

Conditional statements provide a way to execute a chosen code block based on the runtime evaluation of one or more boolean expressions.

Note that, unlike other languages, Python relies on indentation, using 4 spaces, to define code blocks.

loop

python - while loop -

Python utilize two type of loop:
- while
- for

The while loop execute a code block until a certain condition is True

```
>>> i = 1
>>> while i < 6:
...     print(i)
...     i = i + 1
```

WARNING
Verify that there is a condition for exit

python - for loop -

for loop is an instrument useful to iterate on a series of elements. The syntax can be used with an iterable structure (list, tuple, string, set, dict or file)

the list can be iterated with the arbitrary identifier "number"

the block of code will be executed for each element of the iterable object

number → arbitrary identifier containing at each iteration the respective element

```
>>> primes = [2, 3, 5, 7]
>>> for number in primes:
...     print(number)
...
2
3
5
7
>>>
```

python - continue / break -

the keyword break can be used to exit a for or while loop, even if element iteration is not yet finished or while condition is still True

```
>>> i = 0
>>> while i < 7:
...     print(i)
...     if i == 3:
...             break
...
0
1
2
3
>>>
```

the keyword continue is used to pass the current code block and go to the next for or while iteration

```
>>> for letter in 'hombe':
...     if letter == 'b':
...             continue
...     print(letter)
...
h
o
m
e
>>>
```

exception handling

python - exception handling -

When an error occurs (or an exception, as it is called in Python), the code's execution is interrupted, and Python generates an error message. These exceptions can be handled using the try and except clauses.

TypeError: error generated when an operation is applied to an object of the wrong type.
integer + string → raise an error

```
try:
    1 + 'a'
except:
    print("An exeption was raised")
```

The except keyword handles every type of exception. It is possible (and auspicable) to handle the different exceptions differently. To do that is enough to add the exception type after except statement. Most common types are: AttributeError, IndexError, KeyError, NameError, TypeError, ValueError

python - exception handling - else

The try/except statement has an optional else keyword, which, when present, must follow all except keywords. It is useful for inserting code that should only be executed when try does not raise any exceptions.

```
try:
    f = open('test.txt')
except FileNotFoundError:
    print(f"{f.name} not found")
else:
    print("File has been opened")
```

Opening of “test.txt” file attempt

exception handling

If no exception, else block will be executed

python - exception handling - finally

...when you want to execute code regardless of the result of the try/except, you add the finally clause. This code will ALWAYS be executed under any circumstances.

```
# SLIDE NO: 60
try:
    f = open('test.txt', 'x') # try opening file in exclusive creation mode
    f.write("Lorem ipsum dolor sit amet") # writing somethin on the file
except (FileNotFoundError, FileExistsError, PermissionError): # checking for multiple file-related exception
    print(f"{f.name} cannot be opened")
except: # checking for any other exception
    print("Other exception")
else: # executed if no exception arises
    print("File opened and string written")
finally: # in the end closing the file
    print("Closing file")
    f.close()
```

python - assert -

assert is an integrity check that is often placed at the beginning of a function to verify valid input and after a function call to check the validity of the output.

When it encounters an assert, Python evaluates the expression declared after the keyword, which is expected to be true. If the expression is false, Python raises an AssertionError exception.

Syntax → assert Expression [, Arguments]

- I check that the input of the function is an int or a float using the built-in function isinstance();
- custom error message passed as an argument;

```
>>> def square_area(l):
...     assert isinstance(l, (int, float)), "Input Error"
...     return l ** 2
...
>>> square_area(3.5)
12.25
>>> square_area('a')
Traceback (most recent call last):
      File "<stdin>", line 1, in <module>
      File "<stdin>", line 2, in square_area
AssertionError: Input Error
```

AssertionError is raised only with the second function invocation because a string is passed.

python - raise -

The raise statement allows you to raise a specific exception.

```
>>> raise NameError("Raising an exception of NameError type")
Traceback (most recent call last):
        File "<stdin>", line 1, in <module>
NameError: Raising an exception of NameError type
```

If you want to determine if an exception has been raised but don't intend to handle it, simply declaring the raise keyword allows you to re-raise the exception.

```
>>> try:
...             raise NameError("Exception raised")
...     except:
...             print("Exception handled")
...             raise
...
Exception handled
Traceback (most recent call last):
        File "<stdin>", line 2, in <module>
NameError: Exception raised
```

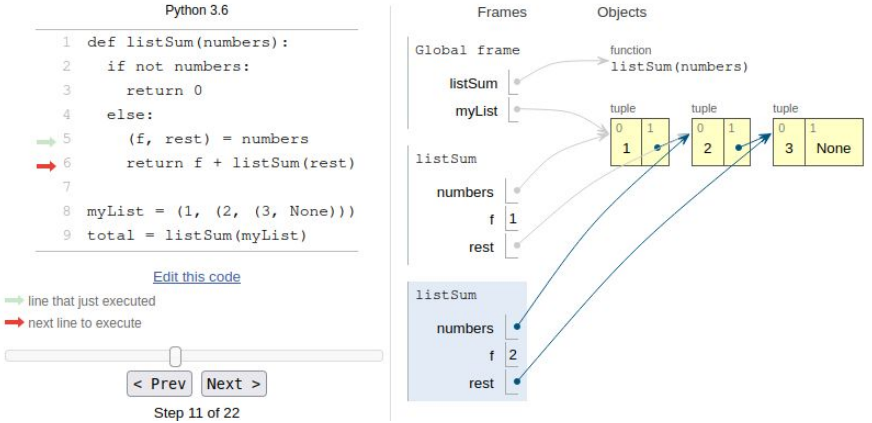
useful tools

useful tools

PythonTutor

https://pythontutor.com/

The website pythontutor.com provides a Python visualization tool that allows you to see how code is executed step by step, making it easier to understand how specific Python constructs work.



tool for training

tool for training

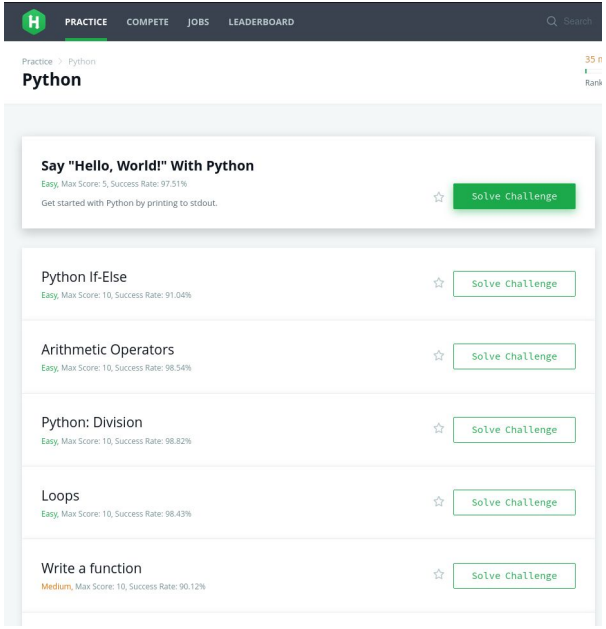
> “Hard work beats talent when talent doesn’t work hard”

A lot of online resources for training:

https://codecombat.com/

https://www.w3schools.com/python/

https://www.hackerrank.com/domains/python



1.part - exercise

exercises - part 1.1 -

input() → function that allows a user to enter a string during the execution of a script [Note: characters entered through input() are ALWAYS converted to a string by Python].

1. Write a script that accepts an input consisting of an integer representing a radius, calculates the area of the corresponding circle, and prints the result.
2. Write a script that accepts three integers. If they are all different, print the sum of the three. If two are the same, print the result of dividing the sum of the equal ones by the third. If all three are the same, print the result of (n + n)n.
3. Write a script that accepts two integers. If the value of their sum, or of their difference is 5, print "True"; otherwise, print "False." If the inputs entered are not numeric, print "Input must be a number."

exercises - part 1.2 -

4. Write a script that accept input for name and surname, and print the string with characters at odd indices in reverse order
5. Write a script that accept an integer 'n', calculate and print the value of n + nn + nn*n
6. Write a script that accept a string and add "I am" to it. If the string already starts with "I am," print the unchanged string
7. Write a script that calculate the difference between a given number and 17. If the number is less than 17, print double the absolute difference (Python has a built-in function called abs(*n*) that return the absolute value of an int or a float).
8. Write a script that for an integer 'n', perform the following conditional operations:
    a. If 'n' is odd, print "bizarre"
    b. If 'n' is even and between 2 and 5 (included), print "not bizarre"
    c. If 'n' is even and between 6 and 20 (included), print "bizarre"
    d. If 'n' is even and greater than 20, print "not bizarre"