Handout 1

Introduction to Python programming language.
Basic data types and arithmetical operations.
Input/output operations.

Python – a general-purpose programming language

- Created in 1990 by Guido van Rossum, in Netherlands
- Interpreted language, originally created as a middle-ground between scripting languages and C
- The core of the standard interpreter is implemented in C
- Quickly gained popularity, volunteers built multiple packages extending language capabilities
- Developed and maintained by volunteers, distributed for free by Python Software Foundation
- **Python 2 and Python 3 are two different versions not compatible with each other.**
  - We will use Python 3, although it is possible to have both versions installed on your computer.

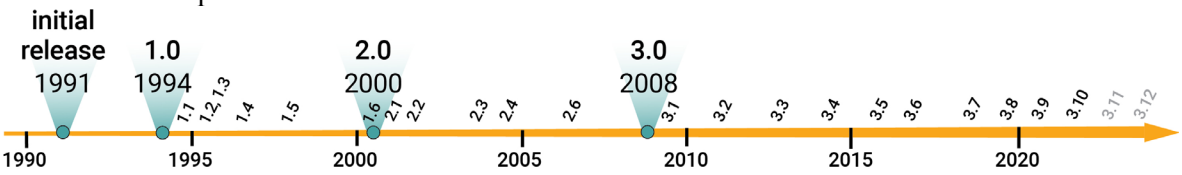
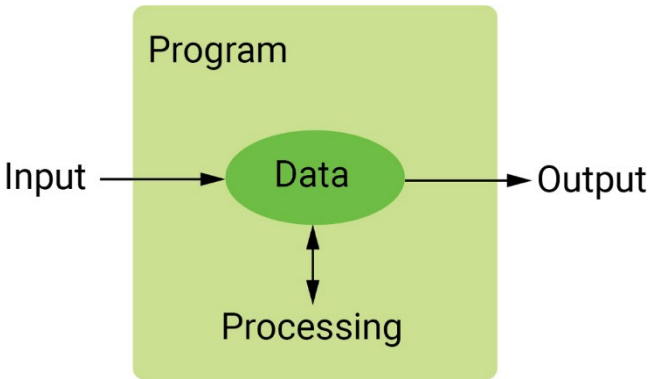


Illustration by Jake VanderPlas

Important features of Python

- Interpreted
- Easily extendable
- Very versatile
- 'easy to use'
- Uses **whitespace** to structure code
- Every data value is an **object**

Programming



FIRST EXAMPLE

```
'''
Example demonstrates
1. getting input as a string and  converting it into a numbers
2. simple output using print
'''

print("Greetings, welcome to CS230!")
dayStr = input("What day is it today?")
print (dayStr)

peopleStr = input("How many people are there in class?")
numPeople = int (peopleStr) # convert string into an integer number

print("There are ", numPeople, "in class on ", dayStr)

print("If one more person comes in, there will be ",
      numPeople + 1 , "people" )
```

Components of the program

Comments: `# for a single line, till the end of line`

```
'''   paragraph-comments
      Start a multiple line comment with three quotations
             End it with three quotations
'''
```

Punctuation and other grammar

- Python is sensitive to white space – indentation is used to indicate nesting of blocks (where other languages use {} or other blog begin-end statements)
  Statements in the same block must be indented in the same way.

- a "**;**" is needed to indicate the end of a statement **only if** there are multiple statements on the same line, e.g.
  `a = 4; b = 5`

- \ is a line continuation symbol

Variable - is a named location to store data

- Python is dynamically typed, meaning a variable type is not declared, it is dynamically inferred based on the value it points to, and a variable can be assigned any type of values at any point.
  - Avoid changing the type of value stored in a variable to avoid errors! Indicate the data type in name for easy understanding

DATA TYPES AND VALUES

Python has a range of built-in types.

Numeric built-in types: `int, float`
have no range limit, i.e. can represent an arbitrary large or small number.

Text built-in type: `str`

```
'''Demo of built-in types'''

    # numeric types: int, float, complex
print (3,     'has type', type(3))
print (-4.5 , 'has type', type(-4.5))

    # have no range limit
a = 300000000000000000000000000000+1
print (a, 'has type',  type(a))
print (1/a, 'has type' ,  type(1/a))

print ('\nStrings -----------------------------------------------------')
     # str type
print (type('abc'))
print (type('a'))
print (type("also a string"))
```

KEYBOARD INPUT

`input('prompt')` – input function, displays prompt specified as a parameter and returns the string of characters entered by the user.

To convert user input from str to another type, use
- Type conversion functions (int(), float()) or eval() function

ASSIGNMENT OPERATOR =

Used to **set, or assign** a variable is to store a given value in the location denoted by the variable name . Not the same as equality in algebra, It means -
"Store the value of the expression on the right side to the variable on the left side."

Can have any expression on the right hand side of =

```
x = 1       # Assign 1 to x
x = x + 1
i = j = k = 1
```

Simultaneous assignment: var1 , var2, ..., varn = exp1, exp2, ..., expn
means: var1 = exp1; var2 = exp2; …varn=expn

```
x,y = 1, 3       # Assign 1 to x, 3 to y
x,y = y, x       # Swap values stored in x and y
```

ARITHMETIC OPERATORS:

`+, -, *, /, //, %, ( ), **`

(1) **integer division**: results from dividing one `int` by another. Returns whole number quotient, ignoring remainder (truncates).

`21 // 4 = ?` `7 // 2 = ?`

(2) **%**: modulus, or remainder.

`21 % 4 = ?` `8 % 2 = ?`

(3) **** -** exponentiation**.**

3** 2 = ?

Practice: what is the value of z in the following?

```
x = 7; y = 3;
z = (x//y)*y + x%y
```

Order of precedence (elements in the same row have the same precedence):

```
()
+(unary)   -(unary)
*          /              //          %
+          -
= (assignment)
```

ROUNDING NUMBERS AND OTHER MATH FUNCTIONS

```
round (numberToRound)
round (numberToRound, decimalDigits )  round (numberToRound, [decimalDigits])
      -  Note, [] denote an optional parameter to a function.

      val =  7.540638
      int(val) returns  7
      round(val) returns  8
      round(val,1) returns  7.5
      round(val,2) returns   7.54

Functions min(), max(), sum() can be used on numbers and sequences.
```

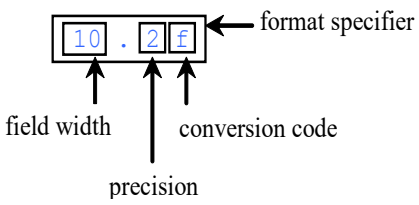
FORMATTING NUMBERS AND STRINGS

format(item, format-specifier) returns a string with **item** formatted according to **format-specifier**

item is a number or a string, format-specifier is a string using special formatting instructions:

`f` float `s` string `d` decimal int `x` hexadecimal int `b` binary int
`%` percentage `<` left-justified `>` right-justified

```
print(format(57.467657, '10.2f'))
print(format(12345678.923, '10.2f'))
print(format(57.4, '10.2f'))
print(format(57, '10.2f'))
```




```
    10
□□□□57.47
12345678.92
□□□□57.40
□□□□57.00
```

```
s1 = 'hello, world'
# center, left, right justify
print (s1.rjust(30))
print (s1.center(30))
print (s1.ljust(30))

# control number of digits AND
#  center, right, left justify
print (format ( 123.5667, '^30f' ))
print (format ( 123.5667, '>30.2f' ))
print (format ( 123.5667, '<30.0f' ))
```

30 characters

```
                  hello, world
          hello, world
hello, world



          123.566700
                                123.57
124
```

A COUPLE OF WORDS ABOUT PRINT()

Function `print()`, by default

- uses a space to separate the components, listed as its arguments,
- places an end of line character to the end of the printed line, so the next print appears on the new line.

To change these two behaviors, you can use optional parameters end and sep

end - defines the ending character
sep – defines the separator between the components.

```
print ('one', 'two', 'three'); print ('four'); →
   one two three
   four
```

```
print ('one', 'two', 'three', sep = "", end = " ")
print ('four')
          →
      onetwothree four
```

STYLE GUIDELINES

Readability and understandability of code are as important as its correctness. The following style guidelines specify how to achieve readability:

Commenting
- Include introductory comments listing the name of the author and a description of the purpose of the program.
- Introduce each major stage of an algorithm in a summary form, so that the reader would not have to read the details code in order to grasp the general idea.
- Comment any code that may be difficult to follow and would benefit from explanatory text.
- Be concise.

Naming and naming conventions
- Use variable names that reflect the purpose of the variable, and add comments if more information would be helpful (for example, if a variable has a valid data range, that should be noted in a comment). Likewise, other names you choose later on for your functions and modules should reflect its purpose. Note that the Refactoring → Rename feature of Eclipse will rename all occurrences of a specified variable,
- Start variable and function names with lower case letters, e.g. `second, degreesCelcius, numPeopleInClass`
- Use all capitals for named constants, e.g. `INTEREST_RATE, PI`, `MIN_WAGE`, etc.

Structuring your code (these requirements will become clear in time)
- Subdivide your program into modules of manageable size (functions, classes, methods) and use white space as appropriate to separate logically separate pieces of code and functionality.
- Simple code is easier to write, debug and understand, therefore it is less prone to error.
- Avoid global variables and do not change the type of value stored in a variable. This will reduce the potential for hard-to-find errors in your code.

DEVELOP THE PROGRAM BY
- identifying the INPUT, OUTPUT and PROGRAM DATA,
- outlining the algorithm and creating test cases,
- implementing the algorithm,
- testing and debugging your code.

PRACTICE PROBLEMS:

1. User will enter number of hours worked by an employee. Assuming the hourly pay rate of $15.50 and 5% tax that must be withheld from the gross amount, compute and display the gross and net amount due to an employee.

2. User will enter the number of people in a class. Your program should calculate and display how many full 4-person teams can be formed and how many people will possibly form a smaller team.

3. A university enrolled 1027 students in 2021 and 1109 students in 2022. Calculate the percentage of the increase in student population from 2021 to 2022.

4. Given the start and end of a class in hours and minutes, compute when a 10 minute break should start and end, if it is supposed to be exactly in the middle of the class. *Hint*: convert all time to minutes to simplify the calculation