Python Basics

James Gain
Department of Computer Science
University of Cape Town
jgain@cs.uct.ac.za

Problem 1 Introduction

- Write a program to print out the following haiku:
- *Type, type, type away*

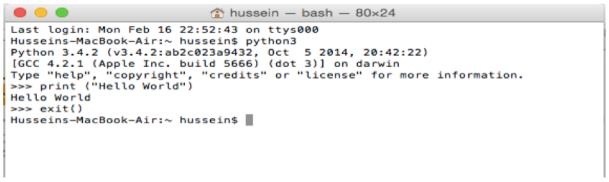
  Compile. Run. Hip hip hooray!

  No error today!

  (By Samantha W.)

Python Interpreter

- Python programs are **run** one instruction at a time by the interpreter.
- We can run the interpreter (usually "python3") and then enter instructions one at a time.
- Python will execute each instruction then **prompt** the user for the next one.
- Enter `exit()` to end.



Integrated Development Environment (IDE)

- Graphical interface with menus and windows, much like a word processor.
- We recommend WingIDE 101
  - Free, scaled-down Python IDE designed for use in teaching introductory programming classes.
- You are welcome to use any other tools – e.g. IDLE
  - Even a separate text editor and interpreter if you like doing things the hard way.

Python program files

- The interpreter is useful for testing code snippets and exploring functions and modules.
- However, to save a program permanently we need to write it into a file and save the file.

- Python files are commonly given the suffix ".py"
  - e.g. `HelloWorld.py`
- Always save your programs while you are working!

Skeleton Python Program

```
# what the program does
# author of program
# date

PythonInstruction1
PythonInstruction2
PythonInstruction3
PythonInstruction4
```

Example Program

helloworld.py:

```
# first program ever
# billgates
# 14 march 1968

print("Hello World")
```

output:

```
Hello World
```

What does it mean?

```
print("Hello World")
```

- **print** is the name of a function.
- "Hello World" is the argument or value sent to this function.
- A function is a common task that we can reuse.
  - Printing on the screen is quite complicated.
  - So we reuse Python's built-in function to make it easier.
- In a few weeks we will also create our own functions.

Problem 1 Introduction

- Write a program to print out the following haiku:
- *Type, type, type away*

  Compile. Run. Hip hip hooray!

  No error today!

  (By Samantha W.)

Problem 2 Introduction

- Write a program to print out your initials in ASCII art.

Program Syntax and Style

- Generally, every statement starts on a new line.
- Case-sensitive
  - STUFF vs stuff vs STuff vs stUFF
- Everything after # is a (for humans) comment.

```
# a simple program
print ("Hi") # write Hi on screen
```

Comments

- Brief description, author, date at top of program.
- Brief description of purpose of each function (if more than one).
- Short explanation of non-obvious parts of code.

```
# test program to print to screen
# James Gain
# 12 Feburary 2019
print ("Hello World")
...
```

Syntax and Logic Errors

- Syntax errors are when your program does not conform to the structure required.
  - e.g., print spelt incorrectly
  - The program will not start at all.
- Logic errors are when your program runs but does not work as expected.
  - You MUST test your program.

Escape sequences

- Escape sequences are special characters that cannot be represented easily in the program.
  - For example ... tabs, newlines, quotes

    \a - bell (beep)

    \b – backspace

    \n – newline

    \t – tab

    \' – single quote

    \" – double quote

    \\ - \ character

Problem 3 Introduction

- Write a program to calculate the number of precious seconds you spend at lectures in a semester, assuming you have 5 lectures a day, lectures on 4 days a week, and there are 12 weeks in a semester.

Numeric Data Types

- For numbers we have two primitive types:
- Integer
  - Whole number with no fractional part
- Floating point number:
  - Number with fractional part
  - WARNING: stores only an approximation to real number
  - There is a limit to the precision, or accuracy, of the stored values
    - e.g. 10/3

Integers: Literals

- Literals are actual data values written into a program.
- Numerical literals can be output just like text, but after sensible conversions:
  - `print (12)`
    `12`
  - `print (12+13)`
    `25`
  - `print (2+2/2)`
    `3.0`

Integers: Expressions

- Common operations
  - + (plus), - (minus), / (divide), * (times), % (mod)
  - // (integer division)
  - ** (a to the power b)
- 11 + 11 // 2 = 16 … how ?
  - Precedence of operators:
    - high: ( )
    - middle: * / %
    - low: + -
  - Left associative if equal precedence.
  - Integer operations when both "operands" are integers.

Integers: Quick Quiz

- What is the value of each expression:
  - `(12 + 34)`
  - `(1 + 2) / (3 - 4)`
  - `5 % 2 + 2 % 5`
  - `1//1//2//3`
  - `((1//2)//3)//4`

Problem 3

- Write a program to calculate the number of precious seconds you spend at lectures in a semester, assuming you have 5 lectures a day, lectures on 4 days a week, and there are 12 weeks in a semester.
- A second/minute * B minute/lecture * C lecture/day * D day/week * E week/semester
- = (A*B*C*D*E) seconds/semester

Problem 4 Introduction

- Write a program to calculate your subminima and final mark for CSC1015F.
- Initialize variables for each component mark at the top of the main method so the marks can be changed relatively easily.

Identifiers

- In source file, *print* is an **identifier**.
- Identifiers are used to name parts of the program.
  - Start with _ or letter, and followed by zero or more of _, letter or digit
  - Preferred style: hello_world, my_first_program
- **Reserved words**:
  - if, for, else, …
  - These identifiers have special meaning and cannot be used by programmers for other purposes.

Identifiers: Quick Quiz

- Which are valid identifiers:
  - 12345
  - JanetandJustin
  - lots_of_money
  - "Hello world"
  - J456
  - cc:123
- Which are good Python identifiers?

Variables

- Variables are sections of memory where data can be stored.
- Most variables have names (identifiers) by which they can be referred.
  - e.g., `a_value, the_total`
- Variables are defined by assigning an initial value to them.
  - `a_value = 10`
  - `the_total = 12.5`

Assignment and Output (I/O)

- Putting a value into a variable:
  - `a = 1`
  - `b = a + 5`
  - `c = 1`
  - LHS is usually a variable, RHS is an expression
- Output values of variables just like literals
  - Can do both on one line
  - `print ("The value is ", a)`

Problem 5 Introduction

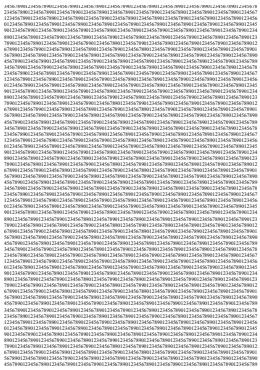
- If we printed out and piled up all the digital data stored on earth how high would it reach?
  - As of 2017
  - Printed out on A4 sheets of paper
  - Stacked on 1/20th of UCT's upper campus
- Write a program to calculate this.

Poll: How high is earth's data?

- Everest (8.84km)
- Suborbital Space (100km)
- The Moon (384,400 km)
- Mars (57.6 million km)

Some Useful Values

- Digital data stored on earth[1]:
  - 2.7 zettabytes in 2017 = 2.55e22 = 25500000000000000000000 bits
  - 1 zettabyte = 1024 ^ 7 * 8 bits
  - A bit (binary digit) is 0 or 1
- Data storage of paper:
  - 6210 characters per page[2] * 6 bits per character = 37,260 bits per page



1. https://www.nodegraph.se/big-data-facts/
2. In Word, with 12pt times roman, no margins

More Useful Values

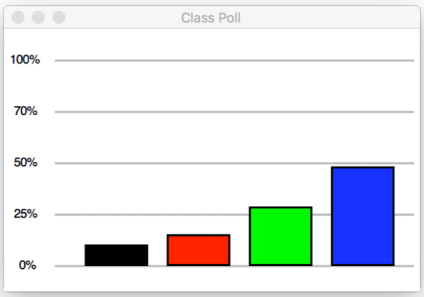
- Area of Upper Campus:
  - 1/20 Red rectangle = 0.05 * 580m * 300m = 8700m^2
- A4 Paper:
  - Area: 210 mm * 297 mm = 0.06237m^2
  - Thickness: 0.075 mm

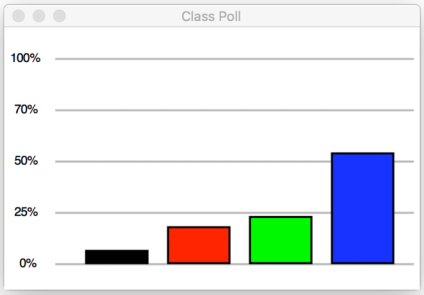
29

- Answer: 367,983km (just short of 384,400km distance to the moon)
- But is this a good idea in practice?
- No
- Number of trees on earth = 3 trillion; Pages per tree = 8,300
- Need ~3 X all the trees on earth

Poll: How high is earth's data?

- Everest (8.84km)
- Suborbital Space (100km)
- The Moon (384,400 km)
- Mars (57.6 million km)





Identifiers

- In source file, *print* is an **identifier**.
- Identifiers are used to name parts of the program.
  - Start with _ or letter, and followed by zero or more of _, letter or digit
  - Preferred style: hello_world, my_first_program
- **Reserved words**:
  - if, for, else, …
  - These identifiers have special meaning and cannot be used by programmers for other purposes.

Variables

- Variables are sections of memory where data can be stored.
- Most variables have names (identifiers) by which they can be referred.
  - e.g., `a_value, the_total`
- Variables are defined by assigning an initial value to them.
  - `a_value = 10`
  - `the_total = 12.5`

Assignment and Output (I/O)

- Putting a value into a variable:
  - `a = 1`
  - `b = a + 5`
  - `c = 1`
  - LHS is usually a variable, RHS is an expression
- Output values of variables just like literals
  - Can do both on one line
  - `print ("The value is ", a)`

Problem 4

- Write a program to calculate your subminima and final mark for CSC1015F.
- Initialize variables for each component mark at the top of the main method so the marks can be changed relatively easily.
  - Final = 0.6 * exam + 0.15 * tests + 0.15 * practicals + 0.10 * practests
  - Theory subminimum = 0.80 * exams + 0.20 * tests
  - Practical subminimum = 0.60 * practicals + 0.40 * practests

Problem 6 Introduction

- Write a program to add two numbers entered by the user and print the result on the screen.

Problem 7 Introduction

- Rewrite problem 4 to get the user to input the marks instead of just storing them in variables.

Output: `print`

- `print` is a built-in function with a precise set of rules for how it works.
- The general format is one of 2 options:
  - `print(<expr1>,<expr2>,…,<exprn>)`
  - `print()`
- By default, print displays the value of each expression, separated by blank spaces and followed by a carriage return (moving to the next line).

Options for `print`

- To print a series of literals or values of variables, separated by spaces:

```
print("The values are", x, y, z)
```

- To suppress or change the line ending, use the end=ending keyword argument. e.g.

```
print("The values are", x, y, z, end='')
```

- To change the separator character between items, use the sep=sepchr keyword argument. e.g.

```
print("The values are", x, y, z, sep='*')
```

Poll: What is the exact output?

```
x=20
y=30
z=40
print("The values are", x, y, '\n', z,
  end='!!!',sep='!***')
```

- ■ `The values are 20 30 40`
- ■ `The values are!***20!***30!***`
  `!***40!!!`
- ■ `The values are 20 30`
  `40`
- ■ `The values are!***20!***30!***`
  `40!!!`

input

- The purpose of the Python input statement is to get a text string from the user; this can be stored in a variable.
- We get input using the input function.
  - `<variable> = input(<prompt>)`
  - prompt is usually some text asking the user for input
  - variable is where the user's response is stored
- For example:

```
weather = input("How is the weather today?")
print ("Weather:", weather)
```

Quiz: What is the exact output?

```
name = input("What is your name? ")
print("Hellooooo",name,end='!!!',sep='…')
```

Inputting numbers

- The input function always gives us a string (text).
- This must be converted into a number if you are going to do math.
  - text (strings) and numbers are handled differently by the computer – more about this later!
- eval() is a Python function that converts a string into a number.
  - `eval("20") -> 20`
- int() and float () also work if we know what kind of number it is.

Sample program

Useful example using eval…

```
# algorithm to calculate BMI, using input
height = input("Type in your height: ")
weight = input("Type in your weight: ")
height = eval(height)  # string to number
weight = eval(weight)  # string to number
BMI = weight/(height*height)  # calculates BMI
print("Body mass index = ",BMI)
```

Problem 6

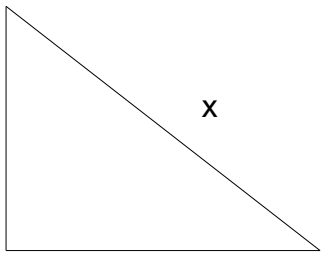
- Write a program to add two numbers entered by the user and print the result on the screen.

Problem 7

- Rewrite problem 4 to get the user to input the marks instead of just storing them in variables.

Problem 8 Introduction

- Use the **math library** for this question.
- Write a program to calculate "x", the length of the hypotenuse on a right-angle triangle, given the other two sides



Other Useful Numerical Syntax

- Increment/decrement operators
- Implicit type conversions
- Explicit type conversions
- math Module

Increment / decrement

- a+=3
  - Equivalent to a=a+3
- Also ...
  - a-=b
  - a*=b
  - a/=b

Implicit Conversions

- Conversion or casting is when one type of value is converted to another
  - E.g., float to integer
- If there is a type mismatch, the narrower range value is promoted up
  -
```
a=1
b=2.0
print (a+b)
```
- Cannot automatically convert down
  - floats do not become ints

Explicit Conversions

- Typecast methods cast (convert) a value to another type.
  - `a = int(1.234)`
  - `b = float(1)`
- Use *math.ceil*, *math.floor*, *round* methods for greater control of floating-point numbers.
- Use *eval (), int(), float()* to convert strings to numbers.
  - `eval("1")`
  - `int("1")`
  - `float("1")`

math Module

- Python has collections of useful functions in **modules**.
- To use a module:
  - `import math`
- Then, to use its functions:
  - `math.sqrt(a)`
- Alternatively, to avoid saying "math.", use:
  - `from math import sqrt`
  - `sqrt(a)`

Sample math functions

Return the base-10 logarithm of *x*. This is usually more accurate than `log(x, 10)`.

`math.`**`pow`**(*x*, *y*)
Return `x` raised to the power `y`. Exceptional cases follow Annex 'F' of the C99 standard as far as possible. In particular, `pow(1.0, x)` and `pow(x, 0.0)` always return `1.0`, even when `x` is a zero or a NaN. If both `x` and `y` are finite, `x` is negative, and `y` is not an integer then `pow(x, y)` is undefined, and raises `ValueError`.

Unlike the built-in `**` operator, `math.pow()` converts both its arguments to type `float`. Use `**` or the built-in `pow()` function for computing exact integer powers.

`math.`**`sqrt`**(*x*) ¶
Return the square root of *x*.

9.2.3. Trigonometric functions

`math.`**`acos`**(*x*)
Return the arc cosine of *x*, in radians.

`math.`**`asin`**(*x*)
Return the arc sine of *x*, in radians.

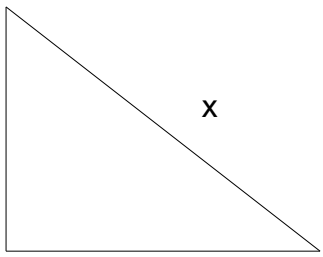
`math.`**`atan`**(*x*)
Return the arc tangent of *x*, in radians.

`math.`**`atan2`**(*y*, *x*)
Return `atan(y / x)`, in radians. The result is between `-pi` and `pi`. The vector in the plane from the origin to point `(x, y)` makes this angle with the positive X axis. The point of `atan2()` is that the signs of both inputs are known to it, so it can compute the correct quadrant for the angle. For example, `atan(1)` and `atan2(1, 1)` are both `pi/4`, but `atan2(-1, -1)` is `-3*pi/4`.

`math.`**`cos`**(*x*)
Return the cosine of *x* radians.

Problem 8

- Use the **math library** for this question.
- Write a program to calculate "x", the length of the hypotenuse on a right-angle triangle, given the other two sides



Terminology

- Comment
  - A part of the code (starting after a # symbol) that is ignored by python and it intended to help a human reader
- Function
  - A named sequence of instructions that perform a collective purpose (e.g., print)
  - Items passed into a function (inside the brackets) are called arguments or parameters
- Identifier
  - A word, consisting of letters, numbers and _ characters, used to name functions and variables.

More Terminology

- Variable
  - A container for data (e.g., strings, integers and floating point numbers), named using the rules for identifiers
- String literal, numeric literal
  - Fixed character sequences or numbers used directly (e.g., "Hello", 3.15)
- Expression
  - An arithmetic expression combining variables, numeric literals, and arithmetic operators (e.g, +, -, etc)
- Syntax
  - The fundamental rules of a programming language
  - Similar to the spelling and grammar rules of natural languages.

Terminology Example

```
# calculate the hypotenuse of a triangle
# James Gain
# 17 February 2019

from math import sqrt
side1 = 3.5 + 7.2
side2 = 4.5
hypotenuse = sqrt(side1*side1 + side2*side2)
print("The hypotenuse is ", hypotenuse)
```

Comment

Expression

Numeric Literal

Function

Variable

String Literal

Parameters

Problem 9

- Write a program to convert your dog's age into human years. Your program must ask for a dog years number and then output the human years equivalent.
- The formula is: 10.5 human years per dog year for the first 2 years, then 4 human years per dog year for each year after.
  - [source: http://www.onlineconversion.com/dogyears.htm]
- Now do it the other way around … human->dog

Problem 10

- Write a program to calculate compound interest, according to the formula:

$$A(t) = A_0 \left(1 + \frac{r}{n}\right)^{n \cdot t}$$

- A_0 = Initial sum, r = Annual interest rate, n = Number of periods per year, t = Number of years
  - Use floating point numbers.