University of
South Australia

Problem Solving and Programming

Week 2 – The Python Standard Library
– User controlled input and output
– Control structures: if statements

Python Books

Course Textbook

Gaddis, Tony. 2012, *Starting Out with Python*, 2nd edition, Pearson Education, Inc.

Free Electronic Books

There are a number of good free on-line Python books. I recommend that you look at most and see if there is one that you enjoy reading. I find that some books just put me to sleep, while others I enjoy reading. You may enjoy quite a different style of book to me, so just because I say I like a book does not mean it is the one that is best for you to read.

- The following three books start from scratch - they don't assume you have done any prior programming:
  - The free on-line book "**How to think like a Computer Scientist: Learning with Python**", by Peter Wentworth, Jeffrey Elkner, Allen B. Downey and Chris Meyers, provides a good introduction to programming and the Python language. I recommend that you look at this book.
  - There is an on-line book "**A Byte of Python**" that is quite reasonable. See the home page for the book, or you can go directly to the on-line version for Python 3, or download a PDF copy of the book. This book is used in a number of Python courses at different universities and is another I recommend you look at.
  - Another good on-line book is "**Learning to Program**" by Alan Gauld. You can download the whole book in easy to print PDF format, and this is another book that would be good for you to look at.
- If you have done some programming before, you may like to look at the following:
  - **The Python Tutorial** - this is part of Python's documentation and is updated with each release of Python. This is not strictly an e-Book, but is book-sized.
  - **Dive into Python 3**, by Mark Pilgrim is a good book for those with some programming experience. I recommend you have a look at it. You can download a PDF copy.

An exercise…

- Assign the value 10 to a variable called width.

An exercise…

- Assign the value 10 to a variable called width.
- Assign the value 5 to a variable called height.

An exercise…

- Assign the value 10 to a variable called width.
- Assign the value 5 to a variable called height.
- Define a variable called area to store the result of width * height.

An exercise…

- Assign the value 10 to a variable called width.
- Assign the value 5 to a variable called height.
- Define a variable called area to store the result of width * height.
- Display the area to the screen.

- Solution…

An exercise…

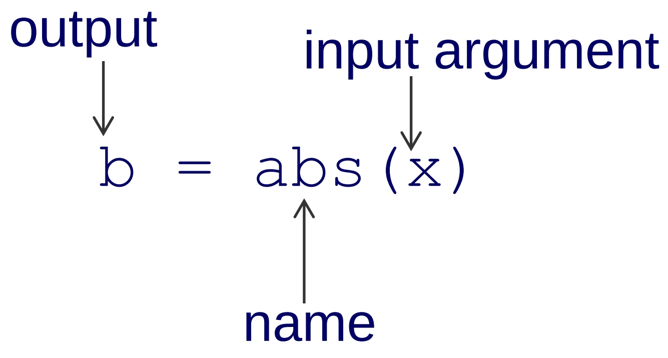
- Assign the value 10 to a variable called width.
- Assign the value 5 to a variable called height.
- Define a variable called area to store the result of width * height.
- Display the area to the screen.

- Solution…

The Python Standard Library

- Built-in functions – extensive library
  - The Python interpreter has a number of functions that are always available.
  - They are available without having to import a library.
  - We will explore a few of the built-in functions.

- Functions consist of three components:
  - Name
  - Input argument(s)
  - Output



```
b = abs(x)
```

- The input (also called argument/parameter) goes inside the parentheses.
- The output is the returned value.

Built-in Python Functions

- Some functions require multiple inputs.

  For example:

  The `pow` function returns x to the power of y.

```
pow(x, y)
```

```
pow(2,3)
```

`pow(2,3)` calculates 2 to the power of 3 = `8`

- You can assign a name to the output (variable):

```
result = pow(2,3)
```

In this case, the name used to refer to the output is the variable name `result`.

Built-in Python Functions

- A few useful built-in functions:

Function	Description	Example

abs(x)	Returns the absolute value of x.	abs(-3)
		ans = 3

pow(x,y)	Returns x to the power y.	pow(5, 2)
		ans = 25

round(x[, *n*])	Returns the floating point value x rounded to *n* digits after the decimal point. If *n* is omitted, it defaults to zero.	round(1.6)
		ans = 2

int([number|string[, base]])	Converts a number or string to an integer. If no arguments are given, returns 0.	int('7')
		ans = 7

float([x])	Converts a string or number to floating point.	float('1.5')
		ans = 1.5

Built-in Python Functions

- A few useful built-in functions:

```
range([start], stop[, step])
```

Constructs progression of integer values, most often used in for loops. The arguments must be integers. The step value defaults to 1 if omitted. If the start argument is omitted, it defaults to 0.

```
str(x)
```

Converts object x to a string representation.

Built-in Python Functions

- A few useful built-in functions:

`print([`*`object, ...`*`], `*`sep=' ', end='\n', file=sys.stdout`*`)`

Print *object*(s) to the stream *file*, separated by *sep* and followed by *end*. *sep*, *end* and *file*, if present, must be given as keyword arguments.

Examples:

```
ans = 7
print("This is an example!")
print("This is another example:", 777)
print(ans)
print("The answer is:", ans)
print("i", "like", "donuts!")
print("i", "like", "donuts!", sep='-')
print("There are", ans, "donuts left...")
```

Built-in Python Functions

- A few useful built-in functions:

`input([`*`prompt`*`])`

If the *prompt* argument is present, it is written to standard output without a trailing newline. The function then reads a line from input, converts it to a string (stripping a trailing newline), and returns that.

Examples:

```
guess = input('Please enter your guess: ')
name  = input('Please enter your name: ')
number = int(input('Enter number: '))
```

Using the Help feature (Python Docs)

- Python has an extensive library of functions.
- We can not possibly teach you all of them (you would go crazy and so would we)!
- Use the help feature to help you use Python’s functions (graduate quality - life long learning).
- Access the help feature from:
  - The Python Shell (IDLE).
  - Select help on the menu bar.
  - Then select Python Docs.
- Standard Library documentation also on the web
- http://docs.python.org/py3k/library/index.html

Numeric and Mathematical Modules

`math` – Mathematical functions

- Provide access to the mathematical functions.
- Need to place the following import statement at the top of your program.

```
import math
```

Numeric and Mathematical Modules

`math` – Mathematical functions

Function	Description
math.ceil(x)	Returns the ceiling of x, the smallest integer greater than or equal to x.
math.floor(x)	Returns the floor of x, the largest integer less than or equal to x.
math.sqrt(x)	Returns the square root of x.
math.cos(x)	Returns the cosine of x radians.
math.sin(x)	Return the sine of x radians.
math.tan(x)	Returns the tangent of x radians.
math.degrees(x)	Converts angle x from radians to degrees.
math.radians(x)	Converts angle x from degrees to radians.
math.pi	The mathematical constant pi.

UniSA

Numeric and Mathematical Modules

`random` – generate pseudo-random numbers

- It provides access to random number generator.
- Need to place the following import statement at the top of your program.

```
import random
```

Function	Description
random.randint(a, b)	Returns a random integer N such that a <= N <= b.
random.choice(seq)	Returns a random element from the non-empty sequence seq.
random.random()	Returns the next random floating point number in the range 0.0, 1.0.
random.shuffle(x)	Shuffle the sequence x in place.

User-Controlled Input and Output

- So far, we have only ‘hard coded’ the values of variables.
- Create more general programs by allowing the user to input values while the program is running.
- The built-in `input()` function allows us to prompt the user to enter a value.

```
value = input('Enter a value: ')
```

- Displays the following text string and waits for the user to provide input.

```
Enter a value:
```

- The value entered by the user is referenced by the variable `value`.
- This is called a prompt because it tells the user to take a specific action.

User-Controlled Input and Output

```
>>> value = input('Enter a value: ')
Enter a value: 7
>>> value
'7'
```

Note: The input function returns a string.

May need to convert this to an integer or floating-point value.

```
>>> value = int(input('Enter a value: '))
Enter a value: 7
>>> value
7
>>> value = float(input('Enter a value: '))
Enter a value: 7
>>> value
7.0
```

User-Controlled Input and Output

Another example:

```
>>> first = input('Enter your first name: ')
Enter your first name: Tony
>>> last = input('Enter your last name: ')
Enter your last name: Stark
>>> name = first + ' ' + last
>>> name
'Tony Stark'
>>> print(name)
Tony Stark
```

User-Controlled Input and Output

- The built-in `print()` function displays information to the screen.

```
>>> a = 3
>>> b = 7
>>> c = a * b

>>> print(a)
3
>>> print(a, b)
3 7
>>> print('a=', a, 'b=', b)
a= 3 b= 7
```

User-Controlled Input and Output

- Another example:

```
print('Hey there...')
name = input('What is your name: ')
print('Hi', name)
print('Nice to meet you...')
print('Let\'s talk again real soon!')
print('Bye')
```

An exercise…

- Let’s revisit the area exercise…

- Modify your program to allow the user to enter a width and height instead of having them fixed to the values of 10 and 5.

An exercise…

- Let’s revisit the area exercise…

- Modify your program to allow the user to enter a width and height instead of having them fixed to the values of 10 and 5.

- Display the area to the screen as follows:

  ```
  The area is:  50
  ```

- Solution…

An exercise…

- Let’s revisit the area exercise…
- Modify your program to allow the user to enter a width and height instead of having them fixed to the values of 10 and 5.
- Display the area to the screen as follows:

```
The area is:  50
```

- Solution…

An exercise…

- Another exercise…

Function	Description
random.randint(a, b)	Returns a random integer N such that a <= N <= b.

- Write a program to generate a random number between 1 – 10.
- Display the random number to the screen as follows:

```
Random number is: 7
```

An exercise…

- Another exercise…

- *Write a program to generate a random number between 1 – 10.*
- *Display the random number to the screen as follows:*

  Random number is: 7

- Modify your program so that it asks (prompts) the user to guess the random number.

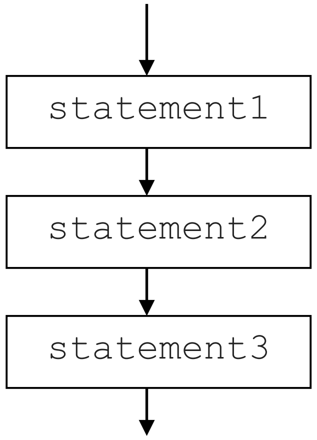
- Display the user’s guess to the screen.

An exercise…

- Solution…

Control Structures

- Control Structures
  - Programs can be written using three control structures (Bohn and Jacopini):
    - **Sequence**
      Statements are executed one after the other in order.
    - **Selection** (making decisions)
      - `if`
      - `if-else`
      - `if-elif-else`
    - **Repetition** (doing the same thing over and over)
      - `while`
      - `for`



Comparison and Boolean Operators

- Comparison/relational operators are used to compare the values of two objects.
- Boolean operators allow us to combine comparisons.
- Comparison/relational Operators
  - Used to make decisions
  - Used in expressions where a True/False result is required.

==	is equal to
!=	is not equal to
<	is less than
<=	is less than or equal to
>	is greater than
>=	is greater than or equal to

UniSA

Comparison and Boolean Operators

- Example:

```
>>> x = 5
>>> y = 1
>>> x < y
False
```

Result of comparison is either `True` or `False`.

The answer may be used in selection statements and repetition statements to make decisions.

Comparison and Boolean Operators

- Boolean Operators (and or not) – combine comparisons.
  - Used when either one, or both, of two (or more) conditions must be satisfied.

```
IF (it is 1pm and I am hungry) THEN
      have lunch

x >= 0 and x <= 5        ⇨ and
x <= 2 or x >= 4         ⇨ or
not a == b and c == d    ⇨ not
```

The expression `x and y` first evaluates *x*; if *x* is false, its value is returned; otherwise, *y* is evaluated and the resulting value is returned.

The expression `x or y` first evaluates *x*; if *x* is true, its value is returned; otherwise, *y* is evaluated and the resulting value is returned.

Comparison and Boolean Operators

- **`not`** negates the value of the operand. That is, converts True to False, and False to True.
- **`and`** requires both p and q to be True for the whole expression to be True. Otherwise, the value of the expression is False.
- **`or`** requires one of p or q to be True for the whole expression to be True. The expression is False only when neither p nor q is True.
- For example:

p	**q**	**p and q**	**p or q**	**not p**
`False`	`False`	`False`	`False`	`True`
`False`	`True`	`False`	`True`	`True`
`True`	`False`	`False`	`True`	`False`
`True`	`True`	`True`	`True`	`False`

Comparison and Boolean Operators

- Can also store the result of an expression.   For example:

```
>>> x = 5
>>> y = 1
>>> z = 5
>>> res = x < y
>>> res
False
>>> res = x == z
>>> res
True
>>> res = y < x and z < y
>>> res
False
>>> res = y < x or y == z
>>> res
True
```

Comparison and Boolean Operators

- Precedence table

Operator	***Description***
`()`	Parenthesis
`**`	Exponentiation
`*  /  //  %`	Multiplication, Division, Remainder
`+  -`	Addition, Subtraction
`<  <=  >  >=  !=  ==`	Comparisons
`not`	Boolean NOT
`and`	Boolean AND
`or`	Boolean OR

Selection Structures

- Making decisions
  - So far, we have seen programs that start at the top and execute each statement in order until the end is reached.
  - We can execute different sections of code depending on the values of data as the program runs.
  - Greater flexibility and therefore more powerful.
  - Let’s look at ways a program can control the path of execution…

Selection Structures

- The `if` selection structure.
- A simple `if` statement has the following form:

```
if expression:
    true_suite
```

- If the expression (boolean expression) is `True`, the statements indented under the `if` statement are executed.
- If the expression is `False`, the program jumps immediately to the statement following the indented block.
- Important: must use consistent indentation – the start and end of blocks are indicated by indentation. Python uses indentation to signify its block structure.

```
if age >= 18:
   print('You can vote!')
```

UniSA

An exercise…

- Let’s revisit the following exercise…
    - Write a program to generate a random number between 1 – 10.
    - Display the random number to the screen as follows:

      Random number is: 7
    - Modify your program so that it asks (prompts) the user to guess the random number.
    - Display the user’s guess to the screen.

- Modify your program so that it displays ‘Well done – you guessed it!’ if the user guesses the number correctly.

An exercise…

- Solution…

Selection Structures

- The `if-else` structure.
- Allows us to execute one set of statements if the expression is true and a different set if the expression is false. An `if-else` statement has the following form:

```
if expression:
     true_suite
else:
     false_suite

if mark >= 50:
   print('You passed!')
else:
   print('You failed!')
```

An exercise…

- Let’s revisit the following exercise…
  - Write a program to generate a random number between 1 – 10.
  - Display the random number to the screen as follows:

    Random number is: 7
  - Modify your program so that it asks (prompts) the user to guess the random number.
  - Display the user’s guess to the screen.
  - Modify your program so that it displays ‘Well done – you guessed it!’ if the user guesses the number correctly.

- Modify your program so that it displays ‘Well done – you guessed it!’ if the user guesses the number correctly, otherwise displays the message ‘Too bad – better luck next time!’ if the user guesses incorrectly.

An exercise…

- Solution…

Selection Structures

- Combining comparisons:

  - **and operator** – true only if temperature is greater than zero and less than 40

```
if temperature > 0 and temperature < 40:
   print('Normal temperature')
else:
   print('Extreme temperature')
```

  - **or operator** – true if either of the conditions are true, that is, if it is raining or highUV

```
raining = True
highUV = True
if raining or highUV:
   print('Take an umbrella!')
```

Selection Structures

- The `if-elif-else` (else-if) structure.
- Nesting `if-else` statements may be difficult to read.
- The `if-elif-else` statement allows you to check multiple criteria while keeping the code easy to read.

```
if mark >= 85:
   print('HD')
elif mark >= 75:
   print('D')
elif mark >= 65:
   print('C')
elif mark >= 55:
   print('P1')
elif mark >= 50:
   print('P2')
elif mark >= 40:
   print('F1')
else:
   print('F2')
```

:
:

An exercise…

- Let's revisit the following exercise…
  - Write a program to generate a random number between 1 – 10.
  - Display the random number to the screen as follows:

    ```
    Random number is: 7
    ```
  - Modify your program so that it asks (prompts) the user to guess the random number.
  - Display the user's guess to the screen.
  - Modify your program so that it displays 'Well done – you guessed it!' if the user guesses the number correctly.
  - Modify your program so that it displays 'Well done – you guessed it!' if the user guesses the number correctly, otherwise displays the message 'Too bad – better luck next time!' if the user guesses incorrectly.

- Modify your program so that it displays 'Well done – you guessed it!' if the user guesses the number correctly, displays 'Too low' if the guess is lower than the random number, displays 'Too high' if the guess is higher than the random number.

An exercise…

- Solution…

An exercise…

- Or…

End of Week 2