INFO1110 / COMP9001 Week 2 Tutorial

Variables, Types and Formatting

Builtin Types

Python has a few built-in types which you can create.

- Integer, **int** - Creates a integer from a series of numeric characters

  ```
  45
  3
  90091
  ```

- Floating point, **float** - Obvious difference is that floating point is denoted with decimal place

  ```
  45.5
  2.2
  12 / 5    //
  ```

- Boolean, **bool** - Boolean variables only have two values associated with them, `True` and `False`

  ```
  a = True
  b = not a #This is False
  ```

- String, **str** - When declaring a string within your code, it is denoted using the ' symbol or " symbol

  ```
  "This is a string"
  'This is a string'
  ```

  `This is not a string` and will cause a syntax error

```
This is not a string -> error

"This is a string"
'This is a string'

'a'
```

Operators

During the lecture you will have been introduced to most of the operators used in Python. This list should provide a quick recap during the tasks of this tutorial and help you get familiar with them.

Arithmetic

- + Add operator, with numerical types (integers and floating point) numbers this adds two numbers while with Strings it will concatenate both strings and create a new string.
- − Subtract operator which will subtract the number from the right hand side from the left hand side.
- ⋆ Multiplication operator, allows two numerical types to be multiplied, this can be used with string types where it will multiply contents of the string N times and produce a new string.
- / Division operator, allows two numbers division of two numbers, division by 0 will result in a 'ZeroDivisionError' **`3 / 0 -> ZeroDivisionError`**

Comparative

- == Equal to, checks for equality between two values
- < Less than, checks a value is less than another value
- <= Less than or equal to, checks a value is less than or equal to another value
- > Greater than, checks a value is greater than another value
- **>=** Greater than or equal to, checks a value is greater than or equal to another value
- != Not equal to, checks for inequality between two values

Logical **`bitwise not ->`**

- `not` Negation, returns a boolean value (True or False) depending on the value. ie if a is true, then **not** a is false
- `and` Logitcal AND, returns True if both values (lhs and rhs) are true, otherwise False
- `or` Logitcal OR, returns True if at least one values (lhs and rhs) are true, otherwise False

Bitwise

- `&` Binary AND operator which will return the value of the bits that only exist in both numbers.
- `|` Binary OR operator, will return the value of the bits
- `^` Binary XOR operator, will return a value where bits exist in both values but if the same bit exists in both values it will exclude it.
- Binary operator, will flip the bits of a value
- « Left shift, will shift all bits left by N
- » Right shift, will shift all bits right by N

```
a = 10011
b = 01000

a ^ b = 11011

~a = 01100

a << 2
a = 10011
a = 011

a = 1011 0011
a << 1
a = 0110 0110
```

a = 0010 1100
a >> 2

Tuples

There is an immutable, aggregate type, called a Tuple. This type can hold multiple values which can be bound to one variable.

Example:

```
x = (1, 'Hello!')
print(x)
(1, 'Hello!')
```

You can expand the contents of a tuple type into separate variables.

```
x = (1, 'Hello!')
y, z = x
print(y)
print(z)
```

Accessing individual elements of tuple uses an indexing from 0. Following this code, we can create a tuple and access each bit of data via an index.

```
x = ('One', "Two", 3)
print(x[0]) # Prints 'One'
print(x[1]) # Prints 'Two
print(x[2]) # Prints '3'
```

Question 1: Naming convention

Int he future, please refer to the PEP8 style guide. Specifically: PEP-8 which outlines naming conventions for variables and functions.

In this exercise, there is a table that lists variable names on the left most column and your task is to determine if a variable is:

- Valid variable name
- Adheres to PEP8 Convention (mainly snake_case)

`age`	
`1st_element`	
`no_times`	
`SIZE_OF_FIELD`	**size_of_field**
`_42`	
`darknessAllAround`	**darkness_all_around**
`superdupervariable`	
`coolplayer2`	
`UserName`	
`_var1_`	

Formatting output

In last week's tutorial we treated printing out integers in a separate statement concatenating the value as part of the string, what would happen if we were to mix both types?

```python
age = 20
print("I am " + age + " years old!")
```

Instead we can create placeholders within a string that can be parsed and evaluated.

```python
age = 20
print("I am {} years old!".format(age)) # New way
print("I am %d years old!" % (age)) # Old way
```

This is a lot more robust and flexible then continually concatenating strings and different data types together to construct a string.

Using C style format strings

Python supports two types of formatting of strings, the old way is using a C-like printf style of formatting strings. The string contains datatype specific placeholders that values are inserted into.

Specifier	Meaning	Options
%d	Signed integer decimal	%4d (padding)
%f	floating point decimaly	%.2f 2 characters after decimal point
%s	String	%-10s, %.5s truncation
%c	Single character	%02 2 character padding

We are not limited to strictly specifying data but we can specify how that data is presented as well, either through truncation, padding or constraining the decimal place on floating point numbers.

```
s = 'Pirates! The Game'
prog = 1/3
times = 4100202

print("Getting: %s, Progress: %.2f, Downloaded: %8d" % (s, prog, times))
```

You may refer to this part of the documentation as how printf-style string formatting works.

printf-style String Formatting

Using format functions (the better way!)

In later versions of python 2, a new way of formatting strings was available which is a lot more flexible and easier to perform. You can still use the classic C-style printf method but we also have the power of a new templating system for data formatting in strings.

Specifier	Meaning	Options
{}	Data placement	.format('data')
{named}	Named data placement	.format(named='data')
{index}	Index data placement	'{1} {0}'.format('one', 'two')
{data[0]}	named data with index or key	'data[0].format(data=d)'

```
s = 'Pirates! The Game'
p = 1/3
t = 4100202

print("Getting: {}, Progress: {.2}, Downloaded: {8}".format(s, p, t))
```

You may refer to Custom String Formatting from the python 3 documentation.

Both styles are very well compared and shown on PyFormat which shows different usage and formats between strings.

Question 2: Using appropriate data types

Although python is a dynamically typed language, you still need to decide the appropriate data representation and storage for the kind of data you are using. Without acknowledging this, your program will encounter strange issues when dealing with

If a light switch is on or off	________________
The time it takes for an olympian to run 100m	________________
Number of rocks on Mars	________________
Percentage of students who ask questions in tutorials	________________
Blood type	________________
Names of the Australian floral emblems	________________
Number of cups of flour to make a cake	________________

Question 3: Boolean expressions

Given the following statements

```
a = True
b = False
x = 50
```

Evaluate the following expressions

`a and b`	________________
`not(not(a)) or b`	________________
`a and not b or not a`	________________
`(b and not(b)) or (a or not(a))`	________________

Question 4: Variable assignment and computation

Given the following program, Annotate each line and show the value of each variable.

```
a = 10
b = 0
b = a
a += a + b
x = a - b
x = a * b
x -= a * b
a = a/2
```

Question 5: State of execution of a program

Given the following program, Annotate and what variables have been declared and what their data type is given their progress of their execution.

```python
a = 3
b = 4
a = '3'
print(a)
j = a + str(b)
print('The value of j is: ' + j)
b += 0.5
o = j
o = len(j) < 2
t = str(o) + j
```

Please show the variables initialised, their type and value at:

- Line 3
- Line 6
- Line 8
- Line 10

Question 6: Type changing of a variable

Discuss with your tutorial group how python would handle changing the data type of a variable? Think about how the interpreter can handle this case and what advantages and disadvantages when programming.

How could we test what the type a variable is and why would we want to do that?

Question 7: Capitalize it all!

Given any line of input to your python program, capitalize all the letters of the string and output it.

```
$ python cap.py
All that glitters is not gold
ALL THAT GLITTERS IS NOT GOLD
```

Question 8: Celsius to Fahrenheit

Write a program that will take celsius as input and convert it to fahrenheit. The output of fahrenheit should only show up to 2 decimal places.

```
python ctof.py
Degrees in celsius: 2.25
Conversion to fahrenheit is: 36.05
```

To convert from celsius to fahrenheit, you are given the formula:

F = C * 9/5 + 32

Question 9: Fahrenheit to Celsius

After finishing the celsius to fahrenheit conversion, write a program that will calculate fahrenheit to celsius.

Question 10: Matches

You are to write a program that will calculate the total number of sets a tennis player has won. You will be provided 2 player names and 5 sets of the games won by each player. If a player has a set where they have won 6 games, they will have won that set, else they have lost.

```
python set.py
Please enter <player1> <player2> <scores>
James Jim 6-5 5-6 2-6 6-4 5-6
James won 2 sets and 24 games
Jim won 3 sets and 27 games
```

Question 11: RGB 24bit Colour

Colour is typically stored as an integer when we talk about monitors, as a programmer, we need to be able to extract and change colour values that are stored as an integer. Bitwise operators such as bit-masking and bit-shifting make this very easy.

How an RGB colour value is stored: Red: First 8 bits Green: 8 bits after Blue: Last 8 bits

Each primary extracted will be within the range of 0-255. You are to also read the number in using hex, a hex value of 0xFFFFFF will correspond to White (255, 255, 255).

```python
#Some scaffold to help
value = int(input('The colour value: '), 16)
red = (value & 0xFF0000) >> 16 #I have amount of red in the colour
#The rest of your code
```

Question 12: Outputting in order

The user will provide a set of comma delimited data. It will be in the form of a name, date of birth pair:

```
name1,dob1
name2,dob2
name3,dob3
```

Use a tuple to store this data and output each person, their name, age and how old they are based on today's current date.

Use Python's datetime module

```python
import datetime
```

To convert a string to datetime:

```python
datetime.strptime(date_variable, '%Y-%m-%d')
```

and use the help(datetime) to bring up documentation with datetime.

Extension - Matches

Extend your set.py program to allow for matches with sets between 3 and 7 and determine a winner at the end.

```
python set.py
Please enter <player1> <player2> <scores>
James Jim 6-5 5-6 2-6 6-4 5-6
James won 2 sets and 24 games
Jim won 3 sets and 27 games
Jim won the match
```

Extension - Is Prime

Write a program to print a message to say whether a given input number is prime. A prime number is one that has just iself and the number 1 as positive integer factors. The first few primes are **2, 3, 5, 7, 11, 13, 17, 19, 23.** The program should print out the first (smallest) factor that divides n.

Here's some pseudocode to help you:

```
if $n \leq 1$ then
    print "not prime"
    return
end if
for $i = 2$ up to $n$ do
    if $i$ is a factor of $n$ then
        print "not prime, i is a factor"
        return
    end if
end for
print "n is prime"
```

Shortcut: actually we only need to check upto square root of n instead of n at Line 4. Why is that?