THE UNIVERSITY OF MELBOURNE

Lecture 2: Variables and Expressions

Dr Simon D'Alfonso

School of Computing and Information Systems
Faculty of Engineering and Information Technology

Today

- In Week 1, we jumped straight into looking at variables and variable assignment in Python, and we started to experiment with the functions print()and input()
- In Week 2, we will look at these things in more depth:
  - understanding the different elements that make up code
  - the naming of variables
  - the importance of data types
  - string expressions
  - arithmetic expressions
  - formatting output
  - the Days of Life calculation - developing a program in parts
- Workshops start this week - Week 2
- Grok

Different elements of code

```
principal = int(input('Enter the initial principal: '))
rate = float(input('Enter the annual interest rate: '))
years = int(input('Enter the number of years: '))
rate = rate/100
amount = principal

for year in range(years):
    amount = amount * (1 + rate)

print('The final value is:', round(amount))
```

- variables, created by programmers
- built-in Python functions
- keywords (reserved words)
- operators
- literals (literal data)

Python Reserved Words, Operators & Built-in Functions

- Python Keywords: https://www.w3schools.com/python/python_ref_keywords.asp
- Python Built in Functions: https://www.w3schools.com/python/python_ref_functions.asp
- Python Operators: https://www.w3schools.com/python/python_operators.asp

Rules for naming variables in Python

- Reserved words cannot be used as variable names (if, for, while, …)
- Do not use built-in functions as variable names (print, input, len, …)
  - print(5)
  - print = 'abc'
  - print(5)
- Names must begin with a letter or underscore (_)
- The rest of the name can contain zero or more occurrences of the following things:
  - digits (0 to 9)
  - alphabetic characters
  - underscores
- Names are case sensitive
  - *HOUSE* is different from *house*

Exercise 1: Variable Name Quiz

- Which of the following are valid variable names?
  - length
  - continue
  - x
  - _width
  - firstWord
  - first_word
  - 2MoreToGo
  - halt!
  - JOURNEY
  - _

Variable Naming Conventions

When choosing names for variables try to be consistent in your approach. Here are some suggestions:

- Be succinct. Keep it short.
- Find the most meaningful word or term that most clearly and unambiguously identifies what the variable represents in the task.
- It can depend on what other variables you need to define: **amount** might be a good name in one program, but confusing in another.
- Use a consistent case type:
  - snake_case (Python convention)
  - camelCase
  - https://chaseadams.io/posts/most-common-programming-case-types/
- Another Python convention is to use UPPER_CASE_SNAKE_CASE for known constants in the task domain, examples being TAX_RATE and STANDARD_DEDUCTION
- PEP 8 -- Style Guide for Python Code
  - https://www.python.org/dev/peps/pep-0008/
  - https://realpython.com/python-pep8/

Exercise 2: Choosing variable names

Write a program that asks the user to input a whole number dollar amount (taking this information into a variable), then adds a GST value of 10%, and then prints the result to the screen.

Exercise 2: Solution

```
VAT = 0.1
amount = int(input('Enter the dollar amount as a whole
number: '))


result = amount + (amount * VAT)


print('The resulting amount with GST applied is $' +
str(result))
```

Data Types

The Basic Data Types

Type of data	Python type name	Examples (literals)
Integers	int	-1, 0, 6895
Real numbers	float	-0.101011, 567738.009187
Character strings	str	"", 'd', "I'm a string", '3'
Boolean	bool	True, False

- The data type of an item determines what values it can hold and what operations it supports
- A literal is a specific or *literal* data value. Variables hold literals.
  - x = 5
  - print(5)
  - print(x)
- You can get the data type of any item (variable or literal) by using the type() function:
  - print(type(x))
  - print(type(5))

String - str

- Strings consist of a sequence of characters, delimited with single quotes (') or double quotes (")
  - 'xI99Pjz_fff2356'
  - 'Hello'
  - '435'
  - "FishPond45"
- Strings include all the alphanumeric letters and numbers, plus special characters - depending on what character set is supported
- Data entered at the keyboard by the user is interpreted first by Python's input() function as a string, because it is entered as a sequence of typed characters

Integers - int

- The integers consist of zero, all positive whole numbers, and all negative whole numbers
  - 34
  - 2
  - -45677
  - 0
  - 1000000
- Integer literals (like the examples above) in Python are written without commas (1000, not 1,000)
- There can be limits to integer representation:
  - A computer's memory places a limit on the magnitude of the largest positive and negative integers
  - The typical int range for Python 2 is/was -2^{31} to 2^{31} (-2147483648 to 2147483647). Numbers beyond that could be stored with the 'long integer' (long) type.
  - Python 3 collapses this distinction, and just has int as large as the system permits.
  - Try evaluating: 2147483647 ** 100

Floating-point (decimal) numbers - float

- Floating-point numbers are Python's way of representing real or decimal numbers, consisting of a whole number, a decimal point and fractional part:
  - 1.45
  - 455.0405
  - 1.00
  - -98.5401
- A floating-point number can be written using either ordinary decimal notation or scientific notation (see next slide)
- Scientific notation is useful when representing very large numbers

Floating-point numbers

DECIMAL NOTATION	SCIENTIFIC NOTATION	MEANING
3.78	3.78e0	3.78×10^{0}
37.8	3.78e1	3.78×10^{1}
3780.0	3.78e3	3.78×10^{3}
0.378	3.78e-1	3.78×10^{-1}
0.00378	3.78e-3	3.78×10^{-3}

Booleans - bool

- There are simply two bool values, **True** and **False**
- Variables can be set to hold Boolean values:
  - x = True
  - y = False
- Boolean values are outputted in Python when a true/false evaluation is made:
  - 2 + 2 == 1 + 3
  - 'abc' == 'abc'
  - 4 > 5
  - 'x' in 'xyz'
  - not False
  - True and False
- You will also use bools when you start using conditionals:

```
if not is_raining and need_exercise:
    go_for_run = True
```

Literals

- Literals are actual pieces of raw data specified in the program code:
  - 'Robert' (a string literal)
  - 45 (an integer literal)
  - 11. 34 (a floating-point literal)
  - True (a Boolean literal)

Exercise 3: Data Types

- Which data types are best to represent the following data?
  1. The number of people who visit a company's website
  2. The average time spent on the website by each visitor
  3. The area of a circle
  4. A password
  5. A company profit
  6. A reason for why a decision was made
  7. A football player's team number
  8. Whether a library book is currently checked out

- Write the values of the following floating-point numbers in Python's scientific notation:
  - 77.89
  - 0.000529

Dynamic typing

- A distinctive feature of Python is that it decides what type a variable is when you first assign a value to it. Take the following example:

```
customer = 'John'
```

- Python creates a variable called 'customer', and then designates it as a string variable, and associates the variable with the literal value 'John'
- This contrasts with *statically* typed languages such as C and Java, in which the programmer is required to explicitly state the data type of each variable when it is being declared in the program
  - int x = 1; (example in C)

Type conversion

- To “cast” a literal value into a desired type, or to cast an existing variable into a different type, we use functions of the same name as the type:
  - int()
  - float()
  - str()
  - bool()
- print(1 + 'a') – what happens when this is run? What is one way to fix this to get the desired output.

Exercise 4: Type Conversion Quiz

- What is the output of the following statements?

1. print(int(34.56))
2. print(int(1.75))
3. print(int(-1.75))
4. print(int('3.45'))
5. print(str(34.56))
6. print(str(10))
7. print(float(4))
8. print(float('3.45'))
9. print(float('abc'))

Exercise 4: Type Conversion Quiz

1. print(bool(1))
2. print(bool(0))
3. print(bool('True'))
4. print(bool('False'))
5. print(bool(7))
6. print(bool(""))
7. print(bool(None))

Expressions

Expressions are fragments of code that produce or calculate new data values:

- 'hello' * 3
- 3 * 3
- x + y
- 'a' + 'b' + 'c'

Escape Sequences

ESCAPE SEQUENCE	MEANING
\b	Backspace
\n	Newline
\t	Horizontal tab
\\	The \ character
\'	Single quotation mark
\"	Double quotation mark

```
name = 'David'
print('Happy Birthday to you \nHappy Birthday to you
\nHappy Birthday Dear ' + name +'\nHappy Birthday to you')
```

Happy Birthday to you
Happy Birthday to you
Happy Birthday Dear David
Happy Birthday to you

Character Sets, and the chr() and ord() functions

- Characters are special strings of length one: 'A', 'a', '9', '*', '@', ' '
- A character-set matches each character to an integer value
- Python strings can contain the following characters:
  - Letters: A to Z, a to z
  - Digits: 0 to 9
  - Special Symbols: + - * / (and more)
  - Whitespaces: Blank Space, tab, carriage return, newline, form feed
  - Other characters: Python can process all ASCII (which represents 128 characters) and Unicode (a much bigger superset of ASCII) characters
- The ord() and chr() functions convert characters to and from their ordinal/number value
- We will see a table of the 128 ASCII characters on the next slide
  - Unicode is a much bigger set that represents a large variety of characters: https://en.wikipedia.org/wiki/List_of_Unicode_characters
  - For those interested, a more detailed primer on encodings: https://realpython.com/python-encodings-guide/

ASCII Character Set

- ASCII control characters (0 - 31)
  - The first 32 characters (and the last) in the ASCII-table are unprintable control codes and are used to control peripherals such as printers.
- ASCII printable characters (32 - 126)

	0	1	2	3	4	5	6	7	8	9
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
1	LF	VT	FF	CR	SO	SI	DLE	DCI	DC2	DC3
2	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
3	RS	US	SP	!	“	#	$	%	&	`
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	‘	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~	DEL		

Exercise 5: Character conversions

- What is the output of the following?
  - print(chr(107))
  - print(ord('H'))
  - print(chr(ord('Q') + 4))
  - print(chr(8712))

Numeric Expressions

Arithmetic Expressions

OPERATOR	MEANING	SYNTAX
-	Negation	-a
**	Exponentiation	a ** b
*	Multiplication	a * b
/	Division	a / b
//	Quotient	a // b
%	Remainder or modulus	a % b
+	Addition	a + b
-	Subtraction	a - b

- The area of a square of sides 7 centimeters: 7**2
- The number of egg boxes that can be completely filled by 33 eggs (with 6 eggs per box): 33 // 6
- The number of eggs left over: 33 % 6

Precedence rules

- Order of evaluation:
  1. parentheses: ()
  2. exponentiation: **
  3. unary negation / negative: -
  4. multiplication, division, remainder: *, / , //, %
  5. addition, subtraction: +, -
- 2 + (3 + 1)**2 / 2 = ?
- With two exceptions, operators of equal precedence are evaluated from left to right (left associative):
  - 7 - 1 + 2 =
- Exponentiation (**) and assignment (=) are evaluated from right to left
  - 2 ** 3 ** 2 = ?
  - x = y = 7
- Use parentheses to force the order that you want AND/OR to make it clear to other humans what the code is doing

unary negation

right associative

EXPRESSION	EVALUATION	VALUE
`5 + 3 * 2`	`5 + 6`	`11`
`(5 + 3) * 2`	`8 * 2`	`16`
`6 % 2`	`0`	`0`
`2 * 3 ** 2`	`2 * 9`	`18`
`-3 ** 2`	`-(3 ** 2)`	`-9`
`(-3) ** 2`	`9`	`9`
`2 ** 3 ** 2`	`2 ** 9`	`512`
`(2 ** 3) ** 2`	`8 ** 2`	`64`
`45 / 0`	`Error: cannot divide by 0`	
`45 % 0`	`Error: cannot divide by 0`	

Exercise 6: What is the output?

- print(10 / 3)
- print (10.0 / 3.0)
- print(10 / 5)
- print(10 // 3)
- print(10.0 // 3.0)
- print(10 % 3)
- print(10.0 % 3.0)

Overloading

- 5 + 7 = 12
- 5 * 7 = 35
- 'a' + 'b' = 'ab'
- 'a' * 7 = 'aaaaaaa'
- + and * means different things for **int** and **str** (and **float** and **bool**). This is a feature of these operators, and operators like this are termed "overloaded", where they function differently depending on the types of the operands.

Formatting outputs

print() - changing the end parameter

- print() statements write a series of expressions to the screen + a default return character:
  - print('one', 'two')
  - print('three', 'four')
  - print()
  - print('five')
- However, the default return character can be changed to another character:
  - print('one', 'two', end=' ')
  - print('three', 'four', end=' ')
  - print('five')

  - print('what is the question', end='?')

f-strings

For more complex output cases, **f-string** is a new Python syntax (from 3.6) that provides a more readable, concise and less error-prone way to format strings than traditional string formatting.

```
>>> name = 'Jane'
>>> age = 26
>>> distance = 17
>>> print(name, 'is', age, 'years old and lives', distance/3, 'kms from the CBD.')
Jane is 26 years old and lives 5.666666666666667 kms from the CBD.

>>> print(f'{name} is {age} years old and lives {distance/3:5.2f} kms from the CBD.')
Jane is 26 years old and lives  5.67 kms from the CBD.
```

- The f' at the start tells Python that this is an f-string
- Everything in {} is evaluated as a variable and the content of the variable is printed as text
- Everything after the colon (:) is formatting instruction – with 5.2f:
  - 5 = character field width
  - For 2f, f denotes floating point and 2 denotes 2 decimal places

Lecture 2 Challenges: Write a program to ...

1. Take four numbers from user input and print their mean (average) value.
2. Take three digits (0 – 9) from user input and print all possible combinations of the digits.
3. Take two numbers from the user, the length of a rectangle and its height (in centimeters), then calculate and print out both the total perimeter of the rectangle and the area.

Lecture Identification and Acknowledgement

Coordinator / Lecturer: Simon D'Alfonso

Semester: Semester 1, 2022

These slides include materials from 2020 - 2021 instances of COMP90059 run by Kylie McColl or Wally Smith