GOVERNMENT OF WESTERN AUSTRALIA

TAFE WA South Metropolitan

Expressions and variables, statements and operators

(Session 2)

Overview

- Fundamentals of computer programming
- Using **print()** function
- Commenting
- Reserved words (also called Keywords)
- Variables
- Operators
- Operator Precedence
- Coding standards

What makes a programming language?

- What is a computer program?
  - It is a set of instructions written to perform some useful tasks
- What is a programming language?
  - It is a language that allows users to write instructions. Programming languages have rules that must be followed:
  - **Alphabet** – a set of symbols used
  - **Syntax** – rules that check that instructions are valid (must be obeyed)
  - **Lexis** – acceptable words
  - **Sematic** – logic in the meaning of instructions (make sense)

What does the interpreter do?

- Interpreter
  - reads the source code
- Editor
  - where you write your code
- Source code
  - It is a program written in a high-level programming language
- Error Messages
  - Interpreter will inform you where the error is located
- Debugger
  - It is used for inspecting and fixing the code

The print() function

- Contains instructions to tell the computer to display the output to the screen. For example, Hello World!
- Note that **string** is delimited with quotes.

```
print("Hello, World!")
```

- The function can contain none or more arguments

```
print()
print("Hello, World!", "More arguments")
```

Using print() function

- The backslash character (****) is called the **escape character**.
  - For example, **\n** means new line, and **\t** means tabs.
  - Note that the next character after the backslash has a different meaning

print("The\t TAB", "adds tabs to the \t string")

```
>>> print("The\t TAB", "adds tabs to the \t string")
The      TAB adds tabs to the    string
```

print("The \nLINE", "adds lines to the \nstring")

```
>>> print("The \nLINE", "adds lines to the \nstring")
The
LINE adds lines to the
string
```

Using print() function

- **sep** – is a keyword argument used to add a character between each arguments
- **end** – is a keyword argument that specifies what character to send when the print() function reaches the end

```
>>> print("sep and end are", "keyword arguments.", sep=":- ", end='**')
sep and end are:- keyword arguments.**
```

Activity 1: experiment with the Python code

print("sep and end are", "keyword arguments.", sep=":- ", end="")**

```
>>> print("sep and end are", "keyword arguments.", sep=":- ", end="**")
sep and end are:- keyword arguments.**
```

Your task: change the *code above* to produce the following output

```
sep and end are = keyword arguments.!!!
```

Printing Quotes and Multi-line strings

- How to add quotes inside of sentence?
- Use **\”**. For example:

print("Using an escape character in front of the \"quotation marks\"")

```
>>> print("use an escape character in front of the \"quotation marks\"")
use an escape character in front of the "quotation marks"
```

- Multi-line strings can be created using three single or double quotes. For example:

print("""Multi-line stings can be

displayed across multiple lines""")

```
>>> print("""Multi-line stings can be
displayed across multiple lines""")
Multi-line stings can be
displayed across multiple lines
```

Activity 2: using print function to create an Ascii art

```
print("""

___ _____ _____ ___
 |  |___ [__   |
 |  |___ ___]  | """)
```

Use this code to produce the output shown below:

```
___  _____  _____  ___
 |   |____  [___    |
 |   |____  ____]   |
>>>
```

Commenting in Python

- To understand what is happening inside your code
- A comment in Python starts with the hash character (#) and whitespace character
- Generally, comments look like this

```
>>> # This is a comment
```

- Because comments do not execute, they can also serve as notes to yourself or reminders

Commenting in Python (cont.)

- You can create **block comments** to explain more complicated code
- You can use a multiline string – that is, triple single ''' or double """ quotes to create a multiline comment
- Python will ignore strings not assign to a value, but not the Python shell

```
#this function will return the value
#of x to the power of y
#for example to return the value of 2 to the
#power of 3 is 2 * 2 * 2
x = pow(2, 3)
```

```
"""This function will return the value
of x to the power of y. """

'''You can also use single quotes
2 to the power of 3 is:
2 * 2 * 2 = 8 '''
```

```
>>> '''this comment is a string literal
with a hidden escape character'''
'this comment is a string literal\nwith a hidden escape character'
```

Variables

- Must have a name and be assign some value (they are created the moment you first assign a value to it)
  - The name of variable can contain the **upper-case or lower-case** letter, **digits** and _ underscore character
  - It **must begin** with letters or underscore (names are case sensitive)
- Naming string variables. For example:

```
str_one = "Variable names"   # using double quote
str_One = 'are case sensitive'    # using single quote
_str1 = "and underscore is a valid identifier"  # valid string name
```

Python Invalid Identifiers - Examples

- **9str1** : identifier starts with digit
- **1111** : can't be only digits
- **!#str1** : can’t use special symbols
- **and** : this is logical operator and therefore not valid identifier

Keywords in Python 3.10 – there are 36 keywords that can't be used as names/identifiers

and	except	lambda	with
as	finally	nonlocal	while
assert	False	None	yield
break	for	not	__peg_parser__
class	from	or	async
continue	global	pass	await
def	if	raise	
del	import	return	
elif	in	True	
else	is	try	

Python Keywords - explained

- Note that all keywords except False, True and None are in lowercase.
- Keywords have special meaning and purposes. For example,
- **if** : used for defining “if’ condition
- **not** : logical operator
- **or** : logical operator
- **and** : logical operator

```
>>> help("keywords")

Here is a list of the Python keywords.  Enter any keyword to get m

False                 break                 for                   not
None                  class                 from                  or
True                  continue              global                pass
```

To get a list of available keywords type:

help("keywords")

help(True) # the help function can be used to define keywords

Basic Data Types in Python

- Variables can store data of different types.
- Text type: **str**
- Numeric types: **int, float**
- Boolean Types: **bool**

- You can get data type of any object using the **type()** function

String Data Type

- String needs quotes
- Each character in the string has a positional value.

str1 = "Hello !"

Character	H	e	l	l	o		!
Index []	0	1	2	3	4	5	6
negative	-7	-6	-5	-4	-3	-2	-1

print(str1[0]) # prints **H**

print(str1[-1]) # prints **!**

```
H
>>>
======
!
>>>
```

Integers ad Expressions

- Integers can contain any whole positive or negative value
- Valid integers:
  - 23, -5, 1, 100
- Can be used in mathematical calculation.
- An **expression** is a combination of values (e.g., 25 + 4)

print(25 + 4)

```
29
>>>
```

Using variables and assigning a new value to an already existing variable

```
var1 = 2            # this is assignment statement
print(var1)         # print statement
var1 = var1 + 2     # assignment with expression
print(var1)         # another print statement
```

```
2
4
>>>
```

Floats

- Have a fractional part after the decimal point
- Valid floats:
  - 2.5
  - -2.5
  - 25.55
  - 0.5
- You can write the value of **0.5** as **.5.** For example,

print(0.5 + .5)

```
1.0
>>>
```

Scientific Notation

- the **exponent** (the value after the E) has to be an integer

```
print(30000.0)
print(3E4)  # the letter E or e means exponent
print(0.0003)
print(3e-4)
```

```
30000.0
30000.0
0.0003
0.0003
>>>
```

Boolean

- Two distinct values (**True** and **False**)
- Used to represent truth values
- In numeric contexts **1** is True, while **0** is False.
- For example,

```
print(True > False)
print(False > True)
```

```
True
False
>>>
```

Arithmetic Operators

- Special characters used to perform operations
- +, -, *, /, //, %, **
- For example, the **+** (plus) is addition operator, – (minus) is subtraction operators, * is multiplication operator, and ** is an exponential operator

```
print(6 + 2)     # outputs 8
print(6 - 2)     # outputs 4
print(6 * 2)     # outputs 12
print(5 ** 2 )   # outputs 25
```

Operators (cont.) and expressions

- A **/** (slash) sign is a divisional operator. The result is always float.
- A **//** (double slash) sign is an integer divisional operator. The result is always integer.
- A **%** (percent) sign is a reminder (modulo) operator.

```
print(6 / 3)     # outputs 2.0
print(6 // 2)    # outputs 2
print(6 % 2)     # outputs 0
```

List of Priorities

Priority	Operator	Higher Priority
( )	Parenthesis	↑
**	Exponent	
*, /, //, %	Multiplication, Division, Integer Division and Modulus	
+, -	Addition and subtraction	

Activity 3: try to work through the following expressions

- print(2 + 5 * 3)
- print(5 * 3 ** 2)
- print((5 * 2) ** 2)
- print(2 ** 2 ** 3)

- Note that expressions in parenthesis are always calculated first.
- The exponential operator uses **right-sided binding**.

Coding standards

- Best practices for code quality
- PEP style guide for Python code. https://www.python.org/dev/peps/pep-0008/
  - 4 spaces per indentation level
  - Spaces are the preferred indentation method
  - Limit all lines to a maximum of 79 characters
  - In Python, single-quoted strings and double-quoted strings are the same. This PEP does not make a recommendation for this
  - Use inline comments sparingly

```
var1 = 2        # this is an inline comment, on the same line as a statement
```

Questions?