| | | |
|---|---|---|
| Faculty of Engineering<br>South Eastern University of Sri Lanka | Course | BSc Eng (Hons) |
| | Semester | 1 |
| | Subject | CS 13001 Introduction to Computing |

SEU/IS/...../EG/.....

Lab Session 2: Variables, expressions, and statements

Lab Objectives:

1) Understanding values and types
2) Understanding variables
3) Learning to distinguish between operators and operands
4) Writing Python expressions and statements
5) Using the 'input' function

## 1. Values and types

A value is one of the basic things a program works with, like a letter or a number. For example: 1, 2, and "Hello, World!" These values belong to different types: 2 is an integer, and "Hello, World!" is a string, so called because it contains a "string" of letters.

***Exercise 1:*** Launch the Python interpreter. Type the commands provided in the first column and write the answers you obtain inside the second column.

| Command to type in the interpreter | Answer |
|---|---|
| >>> print(4) | |
| >>> type('Hello, World!') | |
| >>> type(17) | |
| >>> type(3.2) | |
| >>> type('17') | |
| >>> type('3.2') | |

## 2. Variables

Variable is a name that refers to a value. An *assignment statement* creates new variables and gives them values.

```
>>> message = 'And now for something completely different'
>>> n = 17
>>> pi = 3.1415926535897931
```

This example makes three assignments. The first assigns a string to a new variable named message; the second assigns the integer 17 to n; the third assigns the (approximate) value of π to pi.

***Exercise 2:*** Print out the value and type of each of the variables you assigned above – i.e. message, n, and pi. Use the commands given on the first column.

| Command to type in the interpreter | Answer |
|---|---|
| >>> print(message) | |
| >>> type(message) | |
| >>> n | |
| >>> type(n) | |
| >>> print(pi) | |
| >>> print(type(pi)) | |

### *3. Variable names*

Programmers generally choose names for their variables that are **meaningful** and document what the variable is used for.

Variable names can be arbitrarily long. They can **contain** both **letters** and **numbers**, but they *cannot start with a number*. It is legal to use uppercase letters, but it is a good idea to begin variable names with a lowercase letter.

The underscore character (_) can appear in a name. It is often used in names with multiple words, such as *my_name* or *airspeed_of_unladen_swallow*. Variable names can also start with an underscore character.

***Exercise 3:*** Try the following variable names and identify what is wrong with them.

| Command to type in the interpreter | Why is the syntax invalid? |
|---|---|
| >>> 76trombones = 'big parade' | |
| >>> more@ = 1000000 | |
| >>> class = 'CS 13001' | |

***Exercise 4:*** What are Python keywords? How many keywords are there in total? Give five examples.

***Exercise 5:*** Which set of variable names would you recommend? Tick your choice. Explain why?

```
a = 35.0
b = 12.50
c = a * b
print(c)

hours = 35.0
rate = 12.50
pay = hours * rate
print(pay)

x1q3z9ahd = 35.0
x1q3z9afd = 12.50
x1q3p9afd = x1q3z9ahd * x1q3z9afd
print(x1q3p9afd)
```

### *4. Statements and script*

A **statement** is a unit of code that the Python interpreter can execute. We have seen two kinds of statements: **expression** statements and a**ssignment** statements.

***Exercise 6:*** Give an example of an expression statement?

***Exercise 7:*** Give an example of an assignment statement?

A **script** usually contains a sequence of statements. If there is more than one statement, the results appear one at a time as the statements execute.

***Exercise 8:*** Write the following statements as a python script on Atom text editor. Name the script: first_script.py. Execute the script from the Command Line terminal.

```
print(1)
x = 2
print(x)
```

What is the answer you obtained? Identify the assignment statement from the three statements in the script.

### *5. Operators and operands*

Operators are special symbols that represent computations like addition and multiplication. The values the operator is applied to are called operands. The operators +, -, *, /, and ** perform addition, subtraction, multiplication, division, and exponentiation.

***Exercise 9:*** Try the following in the Python interpreter and note down your answers. Also distinguish between the operators and the operands in the block of statements. Indicate all the variables used in the block of statements.

| Commands to type in the interpreter | Answer | Operator | Operands | Variables |
|---|---|---|---|---|
| >>> x = 20 + 32<br>>>> print (x) | | | | |
| >>> hour = 2.5<br>>>> minutes = hour * 60<br>>>> print (minutes) | | | | |
| >>> square_of_five = 5**2<br>>>> print (square_of_five) | | | | |
| >>> y = (5+9) * (20-2)<br>>>> print(y) | | | | |
| >>> minutes = 59<br>>>> hours = minutes // 60<br>>>> print(hours) | | | | |
| >>> quotient = 7 // 3 | | | | |
| >>> remainder = 7 % 3 | | | | |
| >>> first = 10<br>>>> second = 15<br>>>> print(first + second) | | | | |
| >>> first = '100'<br>>>> second = '150'<br>>>> print(first + second) | | | | |

### *6. Order of operations*

When more than one operator appears in an expression, the order of evaluation depends on the rules of precedence. For mathematical operators, Python follows mathematical convention. The acronym PEMDAS is a useful way to remember the rules:

- Parentheses have the highest precedence and can be used to force an expression to evaluate in the order you want.
- Exponentiation has the next highest precedence, so 2**1+1 is 3, not 4, and 3*1**3 is 3, not 27.
- Multiplication and Division have the same precedence, which is higher than Addition and Subtraction, which also have the same precedence.
- Operators with the same precedence are evaluated from left to right.

***Exercise 10:*** Evaluate the following expressions in the Python interpreter. Explain the answers.

| Expressions to evaluate | Answer | Explanation |
|---|---|---|
| >>> 2 * (3-1) | | |
| >>> (1+1)**(5-2) | | |

| | | |
|---|---|---|
| >>> 2**1+1 | | |
| >>> 3*1**3 | | |
| >>> 2*3-1 | | |
| >>> 6+4/2 | | |
| >>> 5-3-1 | | |

### 7. Asking the user for input

Python provides a built-in function called input that gets input from the keyboard. When this function is called, the program stops and waits for the user to type something. When the user presses Return or Enter, the program resumes and input returns what the user typed as a string.

```
>>> inp = input()
Some silly stuff
>>> print(inp)
Some silly stuff
```

```
>>> name = input('What is your name?\n')
What is your name?
Elijah
>>> print(name)
Elijah
```

```
>>> age = input('What is your age?\n')
What is your age?
50
>>> new_age = age + 10
>>> new_age = int(age) + 10
>>> print(new_age)
```

### 8. Comments

As programs get bigger and more complicated, they get more difficult to read. Formal languages are dense, and it is often difficult to look at a piece of code and figure out what it is doing, or why.

For this reason, it is a good idea to add notes to your programs to explain in natural language what the program is doing. These notes are called comments, and in Python they start with the # symbol:

```
# compute the percentage of the hour that has elapsed
percentage = (minute * 100) / 60
```

In this case, the comment appears on a line by itself. You can also put comments at the end of a line:

```
percentage = (minute * 100) / 60 # percentage of an hour
```

Everything from the \# to the end of the line is ignored; it has no effect on the program. Comments are most useful when they document non-obvious features of the code.

***Exercise 11:*** Which of the following comments is a useful one?

```
v = 5 # assign 5 to v
```

```
v = 5 # velocity in meters/second.
```

# Review Questions

1. Write a program that uses input to prompt a user for their name and then welcomes them.

Enter your name: Chuck

Hello Chuck

2. Write a program to prompt the user for hours and rate per hour to compute gross pay.

Enter Hours: 35

Enter Rate: 2.75

Pay: 96.25

We won't worry about making sure our pay has exactly two digits after the decimal place for now. If you want, you can play with the built-in Python round function to properly round the resulting pay to two decimal places.

3. Assume that we execute the following assignment statements:

```
>>> width = 17
>>> height = 12.0
```

For each of the following expressions, write the value of the expression and the type (of the value of the expression).

1. >>> width//2
2. >>> width/2.0
3. >>> height/3
4. >>> 1 + 2 * 5

Use the Python interpreter to check your answers.

4. Write a program which prompts the user for a Celsius temperature, convert the temperature to Fahrenheit, and print out the converted temperature.