Python Syntax

Data Types, Variables, and Operators

Alwin Tareen

Data Types

Integer data type: `int`

This is any whole number(positive or negative) that does not have a decimal component.

```
average = 9
print(average)
print(type(average))
```

Float data type: `float`

This is any number(positive or negative) which has a decimal component.

```
radius = 28.975
print(radius)
print(type(radius))
```

Data Types

String data type: `str`

- This is any sequence of text characters.
- A string must be enclosed by quotation marks.
- You are allowed to use either the double quotation or single quotation to enclose your string.

```
greetings = "hello"
departure = ‘goodbye’
print(greetings)
print(type(greetings))
```

- **Note**: in this course, we will use the double quotation marks with our strings.

Data Types

Boolean data type: `bool`

- This data type can only have a `True` or `False` value.
- Note that the `T` in `True` must be uppercase.
- Also, the `F` in `False` must be uppercase as well.

```
lightson = True
print(lightson)
print(type(lightson))
```

Some Commonly Used Functions

The `print()` function

This displays to the output whatever value is between the parentheses.

```
print("hello")
print(58)
print(28.975)
print(True)
```

The `type()` function

This indicates the data type of whatever item is between the parentheses.

```
print(type("pizza"))
```

Variable Declaration

What is a variable?

A variable is a named piece of memory that you can use to store information in a `Python` program.

```
total = 36
```

- The name of the variable is: `total`
- The data that is stored in the variable is: 36
- The data type of `total` is: `int`

Note that we did not have to explicitly declare the data type of the variable `total`. `Python` can infer this information automatically.

Variable Naming Rules

1. A variable name must begin with a letter(not a number or symbol).

```
total = 58 # Legal
2scoops = 31 # Not legal
```

2. The variable name must be a sequence of letters or digits, with no spaces in between.
3. Symbols(@, #, $, %, &, etc.) cannot be used in a variable name, with the exception of the underscore character: _

```
good4you = 23 # Legal
work@home = 95 # Not legal
```

4. The length of a variable name is unlimited.

Reserved Python Keywords

- You cannot use a reserved Python keyword as a variable name.
- Python reserves these keywords to perform common programming functionality, such as loops or conditionals.
- The following is a comprehensive list of Python's reserved keywords.

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

Statements & the Assignment Operator

Statements

- A **statement** is a unit of code that the `Python` interpreter can execute.
- A **script** usually consists of a sequence of statements.
- If there is more than one statement, then the results appear one at a time, as the statements execute.

The assignment operator: =

- The equals sign is used to **assign** a value to a variable.
- Be careful not to confuse it with the **mathematical** equals sign, which is used in a different context.

```
total = 58 # assignment
```

Arithmetic Operators

- **Operators** are special symbols that represent computations, like addition and multiplication.
- **Operands** are the elements that the operators are applied to.

Symbol	**Operation**
+	addition
–	subtraction
*	multiplication
/	decimal division
//	integer division
**	exponentiation

Two Different Kinds of Division

Decimal division: /

This performs a normal, calculator-style division, where the result has a decimal component.

- `print(10/3)` $\rightarrow$ 3.333333
- `print(20/7)` $\rightarrow$ 2.857142

Integer division: //

This performs the mathematical floor operation. It performs the division as usual, but the decimal component is discarded(not rounded).

- `print(10//3)` $\rightarrow$ 3
- `print(20//7)` $\rightarrow$ 2

Exponentiation

The exponentiation operator: a**b

This raises the first operand(the base) to the power of the second operand(the power).

- `print(5**2)` $\rightarrow$ 25

The *n*th root

Exponentiation provides an easy way to determine the *n*th root of a number. Simply raise the base to the power of a fraction.

- Square root: `print(25**0.5)` $\rightarrow$ 5
- Cube root: `print(64**(1/3))` $\rightarrow$ 4

Order of Operations

Expressions

An expression is a combination of values, variables, and operators.

Operator Precedence

- When more than one operator appears in an expression, the order of evaluation depends on the rules of precedence.
- In `Python`, the mathematical convention of PEMDAS is used: Parentheses, Exponentiation, Multiplication, Division, Addition, and Subtraction.

Order of Operations

Parentheses

These have the highest precedence, and can be used to force an expression to evaluate in the order that you want.

Exponentiation

This has the next highest precedence.

Multiplication and Division

These have the same level of precedence. Operators at the same level are evaluated from left to right.

Addition and Subtraction

These have the same level of precedence. Operators at the same level are evaluated from left to right.

The Modulus Operator: %

The modulus works on integers, and outputs the **remainder** when the first operand is divided by the second operand.

- `print(7%3)` $\rightarrow$ 1
- `print(25%11)` $\rightarrow$ 3

Checking if a number is even

Perform a modulus operation with 2, and if the result is 0, then that number is even.

- `print(10%2 == 0)` $\rightarrow$ `True`

Checking if a number is odd

Perform a modulus operation with 2, and if the result is 1, then that number is odd.

- `print(17%2 == 1)` $\rightarrow$ `True`

Extracting Digits from a Number

Extracting the right-most digit from a number

Perform a modulus operation with 10, and the result will be the right-most digit.

- `print(98%10)` $\rightarrow$ 8

Extracting the two right-most digits from a number

Perform a modulus operation with 100, and the result will be the two right-most digits.

- `print(1524%100)` $\rightarrow$ 24

Compound Assignment Operators

The += operator

Several assignment operators in `Python` combine a basic operation with assignment. For example, the += operator can be used as follows:

```
score = 10
score += 5 # score is now 15
```

The code above causes the value of `score` to be increased by `5`. The code above is equivalent to the following:

```
score = 10
score = score + 5
```

Python's Compound Assignment Operators

Op.	Description	Example	Equivalence
=	assignment	`x = y`	`x = y`
+=	addition & assignment	`x += y`	`x = x + y`
-=	subtraction & assignment	`x -= y`	`x = x - y`
*=	multiplication & assignment	`x *= y`	`x = x * y`
/=	division & assignment	`x /= y`	`x = x / y`
//=	division & assignment	`x //= y`	`x = x // y`

Collecting Input from the User

- The `input()` function 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 `Enter`, the program resumes execution.
- The `input()` function returns whatever the user typed as a string.
- A programmer can display a prompt to the user, as a guide on what kind of data to input.

```
name = input("What is your name?\n")
print("Hello, " + name)
```

Comments

- It is a good idea to add small notes to your programs, to explain what your program is doing.
- These notes are called comments, and they must begin with the # symbol.
- Any text from the # symbol to the end of the line is ignored by the interpreter.

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

Also, you can write an inline comment as follows:

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

Python Syntax: End of Notes