Introduction to Python

Basic Syntax

Operations and Variables

Topics

1) Operations
   a) Arithmetic
   b) Comparison
   c) Boolean
2) Variables
3) Assignments and Augmented Assignments
4) User input

Arithmetic Operations

Operator	Name	Description
`a + b`	Addition	Sum of a and b
`a - b`	Subtraction	Difference of a and b
`a * b`	Multiplication	Product of a and b
`a / b`	True division	Quotient of a and b
`a // b`	Floor division	Quotient of a and b, removing fractional parts
`a % b`	Modulus	Remainder after division of a by b
`a ** b`	Exponentiation	a raised to the power of b
`-a`	Negation	The negative of a
`+a`	Unary plus	a unchanged (rarely used)

Mixing Types

Any expression that two floats produce a float.

```
In[1]: 17.0 - 10.0
Out [1]: 7.0
```

When an expression's operands are an int and a float, Python automatically converts the int to a float.

```
In[2]: 17.0 - 10
Out [1]: 7.0

In[3]: 17 - 10.0
Out [1]: 7.0
```

True Division vs Floor Division

The operator / is true division and the operator // returns floor division(round down after true divide).

```
In[1]: 23 // 7
Out [1]: 3
In[2]: 3 // 9
Out [2]: 0
In[2]: -4 // 3
Out [2]: -2
In[3]: 6 / 5
Out [3]: 1.2
```

Remainder with %

The % operator computes the remainder from floor division.

- `14 % 4` is `2`
- `218 % 5` is `3`

```
    3
4 ) 14
    12
    2
```

```
   43
5 ) 218
   20
    18
    15
    3
```

- Applications of % operator:
  - Obtain last digit of a number: `230857 % 10` is `7`
  - Obtain last 4 digits: `658236489 % 10000` is `6489`
  - See whether a number is odd: `7 % 2` is `1`, `42 % 2` is `0`

Modulo Operator

The operator % returns the modulus which is the remainder after floor division.

```
In[1]: 18 % 5
Out [1]: 3
In[2]: 3 % 9
Out [2]: 3
In[2]: 125 % 10
Out [2]: 5
In[3]: -17 % 10
Out [3]: 3
In[3]: 17 % -10
Out [3]: -3
```

Note: This is different than Java, which gives the remainder after integer division. But the two are the same for positive operands.

Why floor/modulo division is useful

Floor division allows us to extract the integer part of the division while the modulo operator extracts the remainder part of the division. Consider the question:

How many weeks and days are there in 25 days?

Answer: 3 weeks plus 4 days.

```
In[1]: 25 // 7 # extracting number of weeks
Out [1]: 3

In[1]: 25 % 7 # extracting number of days
Out [1]: 4
```

Why the modulo operator is useful

If today is a Tuesday, which day is 43 days from today?

Answer: 43 divided by 7 is 6 with a remainder of 1. Thus it will be Wednesday.

```
In[1]: 43 % 7
Out [1]: 1
```

Even/odd: A number x is even if x % 2 == 0 and odd if

x % 2 !=0

Expressions

Find the exact change for 137 cents using quarters, dimes, nickels and cents. Use the least number of coins.

How many quarters? 137 / 25 = 5 quarters (Floor Division!)

What's leftover? 137 % 25 = 12 cents

How many dimes? 12 / 10 = 1 dime

What's leftover? 12 % 10 = 2 cents

How many nickels? 2 / 5 = 0 nickels.

What's leftover? 2 % 5 = 2 cents.

How many pennies? 2 / 1 = 2 pennies

What's leftover? 2 % 1= 0 cents. Done!

Extracting Digits

Given a three-digit integer. Extract its the ones, tens and hundreds digits.

For example, if the integer is 352. Its ones digit is the 2, its tens digit is the 5 and its hundreds digit is the 3.

```
number = 352
print("ones:", number % 10)      # ones: 2
number = number // 10            # number = 35 (discards last digit)
print(number)                    # 35
print("tens:", number % 10)      # tens: 5
number = number // 10            # number = 3 (discards last digit)
print(number)                    # 3
```

Note: Modulo 10 (% 10) extracts the last digit. Floor division (// 10) discards the last digit. Later in another lecture, we will see how to generalize this to any number of digits.

Extracting Digits

Alternatively:

```
number = 352
ones = number % 10
tens = (number // 10) % 10
hundreds = number // 100
print("ones:", ones, "tens", tens, "hundreds:", hundreds)
```

Output:

ones: 2 tens: 5, hundreds: 3

Note: You will need the last couple of slides for Replit Classroom labs 2.x.

Exponentiation and Negation

```
In[1]: 2 ** 3
Out[3]: 8
```

Negation is a **unary operator**. It applies to only one operand. Other operations such as +, -, *, /, //, % are **binary operators**, they apply to two operands.

```
In[2]: -5
Out [3]: -5
```

```
In[3]: --5
Out[3]: 5
```

Operator Precedence

Precedence	Operator	Operation
highest	**	exponentiation
	-	negation
	*, /, //, %	multiplication, division, floor division, modulus
lowest	+, -	adding, subtraction

Operators on the same row are applied left to right. Exponentiation, however, is applied right to left. Expressions in parenthesis are evaluated first(PEMDAS).

Operator Precedence

```
In[1]: -2 ** 4
Out[3]: -16

In[2]: 7 - 4 * 5 % (1 + 2)
Out [3]: 5
```

```
7 - 4 * 5 % (1 + 2)
  7 - 4 * 5 % 3
    7 - 20 % 3
      7 - 2
        5
```

Comparison Operators

Operation	Description
a == b	a equal to b
a != b	a not equal to b
a < b	a less than b
a > b	a greater than b
a <= b	a less than or equal to b
a >= b	a greater than or equal to b

Note that = is for assignment and == is for equals.

These operators return either True or False.

Comparison Operators

```
In[1]: 10 == 5
Out[1]: False

In[2]: 3 <= 7
Out[2]: True

In[3]: 3 != 7
Out[3]: True
```

Boolean Operations

Python provides operators to combine the values using the standard concepts of "**and**", "**or**", and "**not**".

These operators are expressed using the words and, or, and not:

X	Y	X or Y
True	True	True
True	False	True
False	True	True
False	False	False

X	Y	X and Y
True	True	True
True	False	False
False	True	False
False	False	False

X	not X
True	False
False	True

Or above is "inclusive or".

Boolean Operations

```
In[1]: x = 4
       (x < 6) and (x > 2)
Out[1]: True

In[2]: (x > 10) or (x % 2 == 0)
Out[2]: True

In[3]: not (x < 6)
Out[3]: False
```

Operator Precedence

Precedence	Operator	Operation
highest	**	exponentiation
	-	negation
	*, /, //, %	multiplication, division, floor division, modulus
	+, -	adding, subtraction
	==,!=,<,>,<=,>=	comparisons
	not	logical not
	and	logical and
	or	logical or
lowest	=	assignment

Boolean Operations

Math operators have the highest precedence. Then comparison operators are followed by logical operators. The assignment operator is evaluated last.

```
In[1]: result = 3 + 2 * 4 < 14 or 3 == 5
       result
Out[1]: True
```

Variables

We can use variables to refer to values that can be used later.

You can create a new variable by given it a value.

```
In[1]: x = 4
       x
Out[1]: 4
```

Variable names can use letters, digits, and the underscore symbol (but they can't start with a digit).

= is not equality

Unlike in math, = is not equality in Python. It is an assignment: assign the expression on the right side of = to the variable on the left.

```
In[1]: x = 4
       x = x + 1   # in math, this has no solutions!
       x
Out[1]: 5
```

Assignment is not symmetric.

```
In[1]: x = 4    # correct!
       10 = y   # error!
```

Augmented Assignment

An **augmented assignment** combines an assignment statement with an operator to make the statement more concise.

Shorthand	Equivalent version
`variable += value;`	`variable = variable + value;`
`variable -= value;`	`variable = variable - value;`
`variable *= value;`	`variable = variable * value;`
`variable /= value;`	`variable = variable / value;`
`variable %= value;`	`variable = variable % value;`

```
In[1]: x = 4
       x += 1   # equivalent to x = x + 1
       x
Out[1]: 5
```

Augmented Assignment

```
In[1]: x = 3
       x *= 2 + 5
       x
Out[1]: 21

In[1]: number = 5
       number *= number
       number
Out[1]: 25
```

Input

Programs may use the input function to obtain information from the user.

```
print('Please enter some text:')
x = input()
print('Text entered:', x)
print('Type:', type(x))
```

Please enter some text:

hello

Text entered: hello

Type: <class 'str'>

Input

Since user input almost always requires a message to the user about the expected input, the input function optionally accepts a string that it prints just before the program stops to wait for the user to respond.

```
x = input('Please enter an integer value: ')
y = input('Please enter another integer value: ')
num1 = int(x)
num2 = int(y)
print(num1, '+', num2, '=', num1 + num2)
```

Please enter an integer value: 4

Please enter another integer value: 5

4 + 5 = 9

Input

Or even more succinctly.

```
num1 = int(input('Please enter an integer value: '))
num2 = int(input('Please enter another integer value: '))
print(num1, '+', num2, '=', num1 + num2)
```

Please enter an integer value: 4

Please enter another integer value: 5

4 + 5 = 9

Lab

Write a program that asks the user to enter three test scores: name these variables test1, test2 and test3. Create three variables: average, variance and standard_deviation and compute their values.

The program then prints out the average, variance and standard deviation. Your program should have a output EXACTLY as below:

```
Enter Test 1 score:78
Enter Test 2 score:80
Enter Test 3 score:77
Average: 78.33333333333333
Variance: 1.5555555555555556
Standard Deviation: 1.247219128924647
```

The variance = $((\text{test1} - \text{ave})^2 + (\text{test2} - \text{ave})^2 + (\text{test3} - \text{ave})^2)/3$

The standard deviation: square root of variance

References

1) Vanderplas, Jake, A Whirlwind Tour of Python, O'reilly Media.

This book is completely free and can be downloaded online at O'reilly's site.