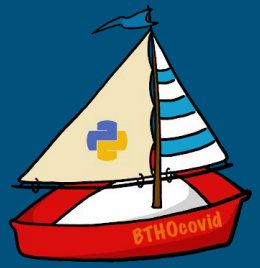


ShiP.py

Learn to Py while Shelter-in-Place

L1: Variables, Expressions, Simple I/O

BY NC SA

A volunteering educational initiative during COVID-19

ShiP Crew

JD

Teddy

Chinmay

Pratik

Siddharth

Umang

Waseem

A volunteering educational initiative during COVID-19

Topics

PHASE I: Foundations

All times are in CDT (GMT-5)

1.	Variables, Expressions, Simple I/O	Sat, April 18 (11 am-12 noon)
2.	Boolean Decisions (branching)	Wed, April 22 (9 pm-10 pm)
3.	Repetitions (loops)	Sat, April 25 (11 am-12 noon)
4.	Collective Data Structures	Wed, April 29 (9 pm-10 pm)
5.	Functions	Sat, May 02 (11 am-12 noon)
6.	File I/O	Wed, May 06 (9 pm-10 pm)
7.	X	Sat, May 09 (11 am-12 noon)

Lecture 1

AGENDA

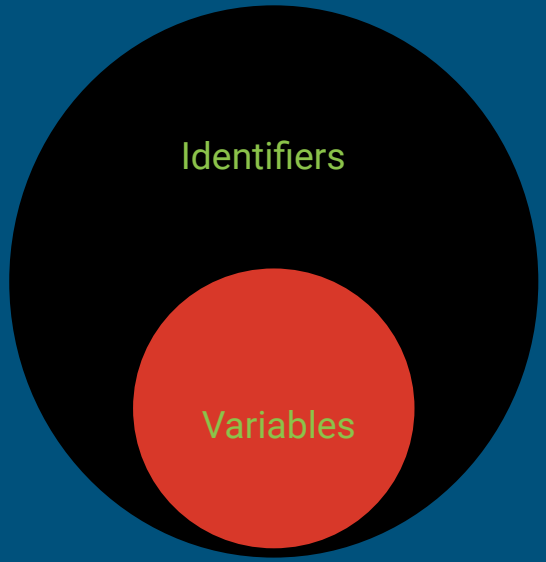
- Identifiers and Variables
- Statements and Expressions
- print( ) function

Identifiers

Names given to identify any python

- Variable
- Function
- Module
- Class

```
grossPay, hours, key
def calculator(), def main()
import math
Class Car()
```

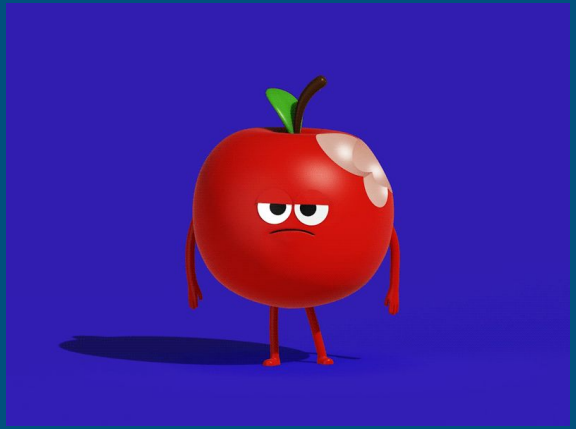


Identifiers: Naming Rules

- Limited to alphanumeric (A-z, 0-9, and _) characters
- May **only** begin with a letter (A-z) or an underscore (_)
- Are case sensitive (*name* and *Name* are two different variables)
- Some words (called keywords) are reserved and cannot be used as identifier

Recommended Conventions/Coding Standards

Keywords

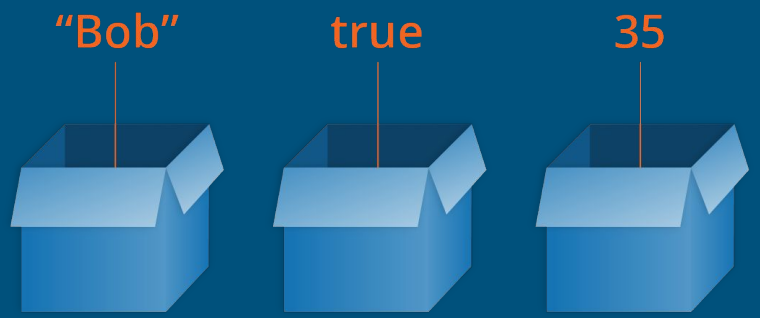


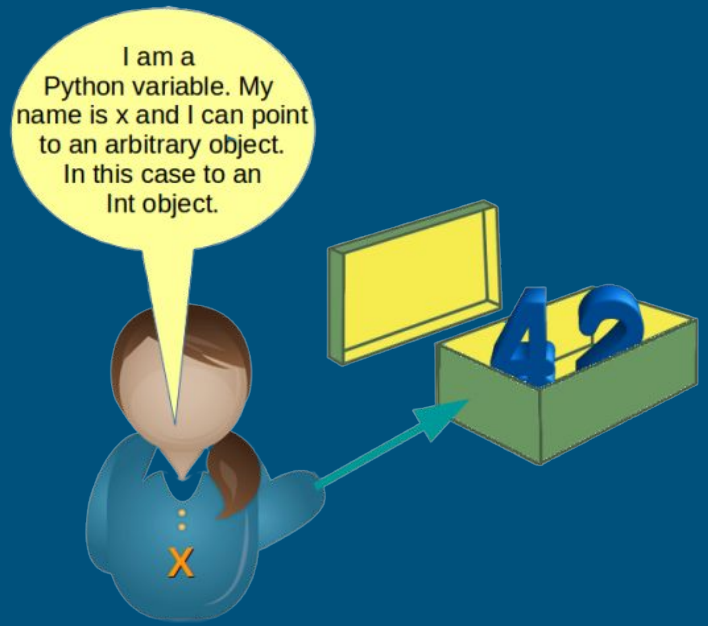
Identifiers

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

Variables

Container to **store** a value in computer's memory





Variables: examples

```python
name = "Harry" # Valid variable
Name = "Ron" # Valid variable, different from 'name' though
print("name has a value of:", name)
print("Name has a value of", Name)
```

```
name has a value of: Harry
Name has a value of Ron
```

```python
# These are all valid variables
num = 100
_num = 10 # Variables can start with underscore
num2 = 201
_num2 = 20.1
# Variables can have numbers between characters / at the end
sh1p_py = "Learn Python while in Shelter-in-place!"
```

```python
# These are all INVALID variables
2num = 100 # Starts with a number
num.2 = 100 # Includes non-alpha-numeric character
try = "I'm invalid" # Is a reserved keyword
```

```
  File "<ipython-input-5-153e4464caf9>", line 1
    2num = 100 # Starts with a number
      ^
SyntaxError: invalid syntax
```

type( ) function

The `type( )` function returns the **type** of specified object within the ( )

```
a = 10
name = 'William'
condition = True
print(type(a))
print(type(name))
print(type(condition)
```

```
<class 'int'>
<class 'str'>
<class 'bool'>
```

Typing: Dynamic and Strong

Dynamic typing: No need to declare variable type. Can be assigned any values. Type of variable is decided during runtime . (unlike C++, JAVA, etc)

Strong typing: Values **can't** change type implicitly.

`variable = 1  # int` `variable = "howdy"  # str`	Variables can be reassigned, even into other types
`print("howdy" + 1)`	TypeError (strong typing)
`print("howdy" + str(1))`	howdy1

User input and Type Casting

Casting to specific type of variable

Welcome message (optional)

```
payRate = float(input("Enter the hourly pay rate: "))
```

Input function

- input() as the name suggests, allows for user input
- Input() by default takes in user values as string type,i.e. str
- Casting function, here, float() is used to specify the desired variable type to store into

Statements and Expressions

Expression

Operator

```
grossPay = hours * payRate
```

Statement

Statements

Any valid line of code

```
age= 35                          #assigning int

name = 'John Smith'              #assigning str

grossPay = hours * payRate       #calculation

print("This is a calculator")     #print statement

if(key == '+'):                  #if condition

def calculator(a,b,key):          #calculator function
definition

calculator(10, 12, '-')           #calculator function call
```

Expressions

Anything that evaluates a value

Numbers	`20, 1.5, 202, -34`
String	`'John'  'Karen'  'Milk'`
Boolean	`True, False`
Calculation	`a+b, 10+20,  20//4`

Operators

https://www.w3schools.com/python/python_operators.asp

Arithmetic Operators

Operator	Meaning/ Result	
+	Adds two numbers	20+35 #55
-	Subtracts two numbers	52-32 #20
*	Multiplies two numbers	3*10 #30
/	Division (Floating point)	5/2 #2.5
//	Floor (or Integer) Division	5//2 #2
%	Modulo Operator. Returns remainder after division	5%2 #1
**	Exponentiation. Returns first value raised to power of second value	3**3 #27

Relational Operators

Return a boolean value, either **True** or **False** and not both

Syntax: value1 *operator* value2 eg: **a > b**

Operator	Meaning/ Result (a *operator* b)
==	Equal to. Returns True is a is equal to b
!=	Not equal to. True if the values are not equal
<	Less than. True only if a is less than b
>	Greater than. True only if a is greater than b
<=	Less than or equal to. True is a is less than or equal to b
>=	Greater than or equal to. True if a is greater than or equal to b

Operator Precedence

When there are multiple operators in an expression, which operation should be performed first ?

```
20+ 15 * 4
```

If + is considered first	If * is considered first

✗ `20+15*4`
`= 35*4`
`= 140`
	✓ `20+15*4`
	`= 20+60`
	`= 80`

Like Math, Python also has a set of rules which instructs which operator to consider first!

Highest

Lowest

Operator	Description
`( )`	Parenthesis
`**`	Exponentiation
`*, //, /, %`	Multiplication, Division
`+,-`	Addition, Subtraction
`==, !=, >, <, >=, <=`	Relational Operators
`not`	Boolean NOT
`and`	Boolean AND
`or`	Boolean OR
`=`	Assignment

To be certain of operator precedence, use parenthesis ( ). This will ensure that value inside ( ) is evaluated first. Eg: **a ** b / (c-d)**

Augmented assignment

Shorthand for updating value of Variables

```
#add 15 to a and save result as a
a=a+15
```

Is same as

```
#add 15 to a and save result as a
a+=15
```

Normal way	Augmented way
`num = num-1`	`num-=1`
`c = c/2`	`c/=2`
`b = b*5`	`b*=5`
`a = a**2`	`a**=2`

Python does not support post and pre-increment/decrement operators
Eg: `a++, --a`

print ( ) function

```
#SYNTAX
print(object(s), sep='separator', end='end')
```

Parameter	Description
`object(s)`	Any object (0 or more)
`sep = 'separator'`	**Optional.** To specify how to separate multiple objects passed on to the function. Default value is one space ' '
`end = 'end'`	**Optional.** To specify how to end the print statement. Default value is new line '\n'

print( ) examples

```
print()

print("This is a calculator")

print("The gross pay is = $",grossPay)  # assuming grossPay is defined

print(a+b)     # assuming a and b are defined

print("Difference = ",a-b)     # assuming a and b are defined

print(a, b, “howdy world!”)   # assuming a and b are defined

print(calculator(a,b,key))     # assuming a, b, key, and calculator() are defined
```

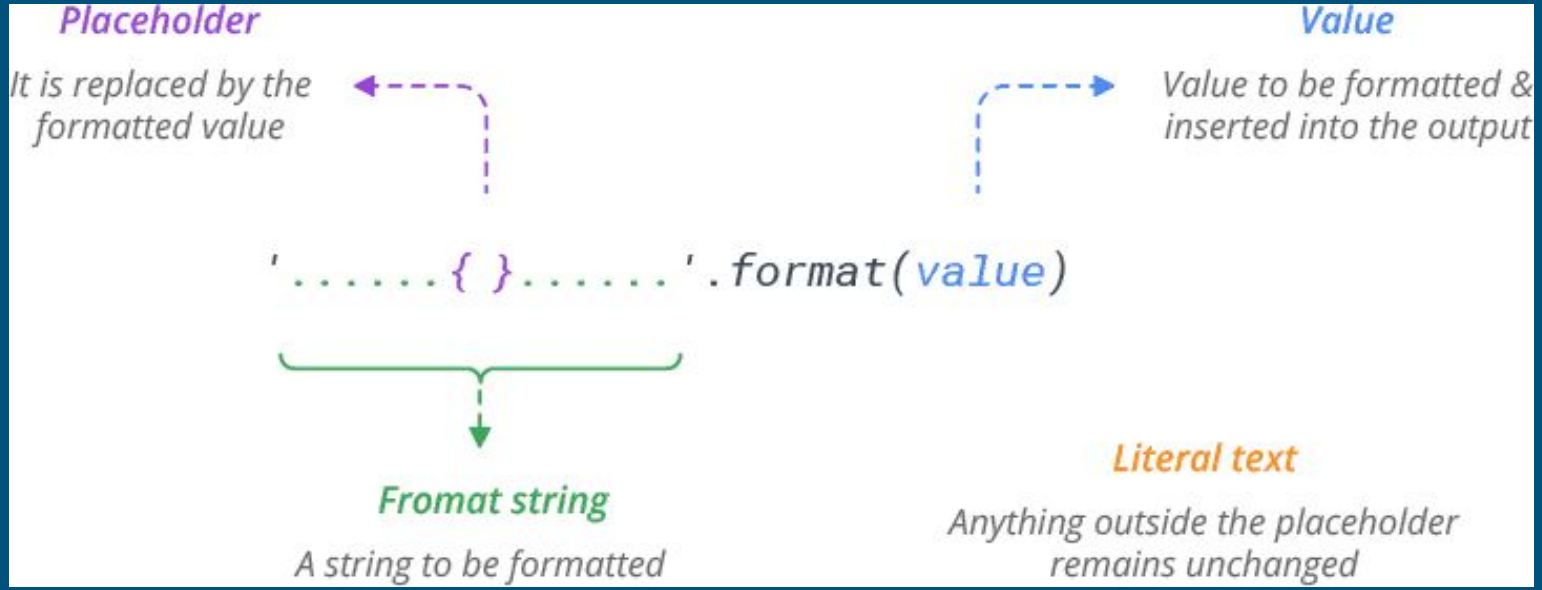
What if you want to print variable values inserted within the String ?

```
My name is John Smith, I am 25.
```

```
Good job Harry, 20 points to Gryffindor.
```

Formatting strings using .format( )

Most commonly used to insert the value of variables inside the string



Source

Placeholder for .format( ) function

{ } is used to reserve a spot for a ***variable*** in the string

`{variable_name}`	Named index or variable
`{0}`	Numbered index
`{ }`	Empty placeholder

.format( ) Examples

```
print("My name is {fname}, I am {age} !".format(fname = "John Smith", age = 25))
```

```
My name is John Smith, I am 25!
```

```
print("My name is {0}, I am {1}! ".format("John Smith",25))
```

```
My name is John Smith, I am 25!
```

```
print("My name is {}, I am {}!".format("John Smith",25))
```

```
My name is John Smith, I am 25!
```

f-string (formatted string literal) in Python 3.6

A shorthand for writing formatted strings to insert variables inside the string

```
fname = "John Smith"
age = 25
print(f"My name is {fname}, I am {age}!")
```

```
My name is John Smith, I am 25!
```

You can also use valid python expressions within the { } in *.format( )* and *f-strings*

```
print(f"2 times 30 is {2*30}")
```

Formatting using %s, %d placeholders

`%s` – for string object

`%d` – for integer variables

`%f` – for floating-point variables (real numbers)

```
print("My name is %s, I am %d." % ("John Smith", 25))
```

```
My name is John Smith, I am 25.
```

Python's `.format()` function is faster and better than `%` operator.
There is much more to `.format()` than stated here. Details

Next Lecture,

L2: Boolean Decisions (Branching)

Wed, April 22 (9 pm-10 pm CDT)