PYTHON SYNTAX, DATA TYPES I

VARIABLES, IDENTIFIERS AND RESERVED WORDS

VARIABLES AND STATEMENTS (INSTRUCTIONS)

ASSIGNMENT STATEMENTS

VARIABLES AND DATA TYPES

EXPRESSION AND OPERATORS

CONTENT

Reserved words (Can’t be used to name variables)

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

snake_case or camelCase

Can’t contain special characters !@#$%^&*()+

Can’t start with a capital letter

Must be clear and make sense, no single letters (be descriptive)

VARIABLE NAME REVIEW

Each variable has a data type.

String: str A series of characters: “A string” or ‘a string’

Integer: int A whole number: 3 or 59

Float: A floating number, a decimal point: 3.4 or 1.0 or 5.224

Boolean: bool A true or false value

Complex: A set of numbers (real and imaginary): complex(3,5)

List / Array: a list of items [10, 30, 5, 30]

PYTHON DATA TYPES

You can check the type using type(varName). Here, I am making a variable and printing the result of type.

```
myIntExample = 10
print(type(myIntExample))
>> <class 'int'>

myFloatExample = 10.0
print(type(myFloatExample))
>><class 'float'>

myStringExample = "10"
print(type(myStringExample))
>><class 'string'>

myListExample = [10, 10, 10]
print(type(myListExample))
<class ‘list’>

myBoolExample = True
print(type(myBoolExample))
<class 'bool'>
```

CHECK TYPE

Input always produces a string, but we can cast it to another data type.

```
adjacent = float(input("enter adjacent side: “))
adjacent = int(input(“enter adjacent side: "))
```

INPUT AND DATA TYPE

This is showing how when the input comes in, it is a string, but we can cast it to a float

```
adjacent = input("enter adjacent side: ")
print(type(adjacent))
# prints <class 'str'>
adjacent = float(adjacent)
print(type(adjacent))
# prints <class ‘float’>

Or
adjacent = float(input("enter adjacent side: “))
You can also assign it at input, or when it is used. It depends
what you are doing.
```

CAST

```
myVariable = 3
# Assign a number
myVariable = 10+5
# Basic expression
myVariable = anOtherVariable
# Assignment to a variable
myVariable = anOtherVariable+5
# Assign expression with a variable
```

ASSIGNMENT

Addition: + myVariable = 10+5

>> 15

Subtraction: - myVariable = 10-5

>> 5

Multiply: * myVariable = 10*5

>> 50

Division: / myVariable = 10/5

>> 2

Modulus: % Modulus is the remainder of division myVariable = 10%5

>> 0

Exponent: ** myVariable = 10**5

>> 100000

ARITHMETIC OPERATORS

```
howManyMinutes = float(input("how many minutes did it take? “))

hours = howManyMinutes / 60

print("it took",hours,"hours")
```

BASIC CALCULATIONS

```
if condition:
    #do something
else:
    #do something else
```

If statements can make decisions. If the statement is TRUE, it will execute the code that is indented. Be careful to look at the indentation and verify.

Don’ t forget the : at the end of the statement. We can use comparison and logical operators here.

IF STATEMENTS

Is equal to ==

Is not equal to !=

Is greater than >

Is less than <

Is greater than or equal to >=

Is less than or equal too <=

COMPARISON OPERATORS

```
myNum = int(input("Can you guess my number? “))
secret_number = 6

if myNum == secret_number:
    print("yeah thats it!”)
```

You can add an else statement if you want, but you don’t need one. The if statement knows to end when the indentation is over, so be mindful of indentation.

```
if myNum == secret_number:
    print("yeah thats it!")
else:
    print("no, but thanks for playing")
```

IF STATEMENTS

```
if (4*4) < (4*6):
    print("4*6 is greater")
else:
    print("This condition will never be met”)
```

If you are comparing two expressions its a good idea to use brackets so that you are sure the proper result is used to make the comparison

IF STATEMENTS

```
if (4*4) < (4*6):

if 4*4<4*6:
For example, its easier to read and ensures correct order.

In this example, the result is different depending on the use of parentheses
print(1 + 2 * 3)

print((1 + 2) * 3)
```

Operations in parentheses () are resolved first, moving from left to right.

** is resolved second, moving from left to right

* / and //and % are resolved third, moving from left to right.

+ and - are resolved fourth, moving from left to right.

ORDER OF OPERATIONS

IN to check if x is IN x

NOT IN to check if x is NOT in y

MEMBERSHIP OPERATORS

```
myAnimal = input("Do I have a dog or a cat? “)
if "cat" in myAnimal:
    print("Yes, I have a cat!”)
else:
    print("incorrect")
```

STRINGS

```
animals = ["cat", "dog", "frog", "turtle"]
my_animal = input("do you have a pet?")

if my_animal in animals:
    print("you have a real pet")
else:
    print("your pet is fake")
```

We will discuss lists next week, but you can try now. Lists are a way to store lots of information in the same variable. We can use the in operator to search the entire list.

```
print(animals[0]) # prints cat
```

STRINGS & LISTS

```
if "cat" in myAnimal.lower():
```

There are a series of methods useful for strings that can help you assess the content of a string. In this example, we can use .lower() on the string to make it all lower case for the comparison. This will be useful if the user types “CAT". .lower() makes it into “cat” so that it can be compared

All string methods are here: https://www.w3schools.com/python/python_ref_string.asp

We will look in detail next week

STRING METHODS

```
if condition:
    # if the first condition is met
elif condition:
    # if not, then if the second condition is met
else:
    # if anything else
```

You can add as many elif statements as you want, but only ONE will be executed. If none are true, else will run. They will be checked in order.

IF ELIF ELSE